HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

DIPLOMARBEIT

Herr
Christian Schikora

Vergleichende Untersuchungen von integralen Längensensoren nach dem Frequenz- und Phasentrackingverfahren

2011

Fakultät **Informations- und Elektrotechnik**

DIPLOMARBEIT

Vergleichende Untersuchungen von integralen Längensensoren nach dem Frequenz- und Phasentrackingverfahren

Autor:
Christian Schikora

Studiengang:
Elektrotechnik, Kommunikationstechnik

Seminargruppe:
ET05wK1

Erstprüfer:
Prof. Dr.-Ing. habil. Heinz Döring

Zweitprüfer:
Dipl.-Ing. Werner Mothes

Leipzig, Februar 2011

Bibliographische Beschreibung:

Christian Schikora:
Vergleichende Untersuchungen von integralen Längensensoren nach dem Frequenz- und Phasentrackingverfahren - 2010, - 29 Seiten, Leipzig, Hochschule Mittweida, Diplomarbeit, 2010

Referat:

Der Schwerpunkt dieser Diplomarbeit liegt auf dem Vergleich von Frequenz- und Phasentracking für ein Längensensor bei identischen Randbedingungen. Dafür ist ein Modul für das Modulsystem der Lehr- und Forschungsgruppe Optronik zu erstellen und zu testen. Dies beinhaltet auch die Erstellung einer Mikrocontroller- als auch einer Hostsoftware, welche ebenfalls zu dokumentieren ist. Ziel ist es, Schlüsse über das zu bevorzugende Verfahren zu ziehen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

µC	Mikrocontroller
ASCII	American Standard Code for Information Interchange
ASK	Amplitude Shift Keying
CLK	Clock
COM	Communication Equipment, Synonym für die RS232-Schnittstelle
CS	Channel Select
DAC	digital-to-analog converter
DDS	Direct Digital Synthesis
FSK	Frequency Shift Keying
I/O	Input/Output
LSB	Least Significant Bit
MISO	Master in Slave Out
MOSI	Master Out Slave In
MSB	Most Significant Bit
PLL	Phase Locked Loop
PoE	Power over Ethernet
POF	polymere optische Faser
PROM	Programmable Read Only Memory
PSK	Phase Shift Keying
SDIO	Serial Data I/O
SPI	Serial Peripheral Interface
USART	Universal Synchronous/Asynchronous Receiver Transmitter
XML	Extensible Markup Language

1 Einleitung

1.1 Motivation

Nachgebender Untergrund in Folge von Instabilität, Setzungsvorgängen oder Ausspülungen stellen eine große Gefahr für die Statik von Bauwerken und direkt oder auch indirekt für Menschenleben dar. Da es auch Jahrzehnte nach dem Bau von Gebäuden zu Veränderungen im Untergrund kommen kann, ist insbesondere bei kritischen Bauwerken wie Talsperren eine Überwachung der Stabilität erforderlich. Aber auch Erdrutsch gefährdete Gebiete wie beispielsweise an Küsten oder Tagebaurestlöchern erfordern zum Schutz von Anwohnern eine Überwachung. Das von der Lehr- und Forschungsgruppe Optronik entwickelte Modulsystem soll dabei ein einfaches, kostengünstiges und zuverlässiges Sensorsystem zum Erfassen von Setzungsvorgängen in Bauwerken oder Abhängen werden.

Prinzipiell besteht das Modulsystem aus einer Messfaser, die als Sensor fest im zu überwachenden Bauwerk oder Untergrund eingebracht wird und auf die Längenänderungen einwirken, einer Messelektronik als Schnittstelle, in der die Signale für den Messfasereingang erzeugt und am Ausgang erfasst werden und einem Messrechner, welcher die Messungen überwacht und steuert. Mit Hilfe des Modulsystems können verschiedene Messmethoden und Bauelemente aufgebaut, getestet und verglichen werden. Ziel dabei ist es, ein Optimum aus geringem Aufwand, niedrigen Kosten, erreichbarer Genauigkeit und maximaler Zuverlässigkeit zu erreichen. Ebenso soll ein großer Kenntnisumfang individuelle Anpassungen und Lösungen ermöglichen. Zahlreiche Module wurden bereits erstellt und getestet. Von den beiden möglichen Trackingverfahren zur Längenänderungsdetektion wurde bisher jedoch nur das Frequenztracking berücksichtigt.

1.2 Aufgabe

Um bei dem Modulsystem eine genaue Messung durch einen Regelkreis zu ermöglichen, ergeben sich, da die Messung der Längenänderung auf der Messung der Phasenlaufzeitveränderung eines Signals durch einen Wellenleiter beruht, zwei Möglichkeiten der Nachführung. Diese Diplomarbeit hat das Ziel, die zwei möglichen Trackingverfahren, Phasen- und Frequenztracking, miteinander zu vergleichen. Zu diesem Zweck ist ein Modul mit zwei synchronen Frequenzgeneratoren, welches die Nutzung beider Verfahren ermöglicht, aufzubauen, in Betrieb zu nehmen und anzusteuern. Dies beinhaltet auch die Programierung des Mikrocontrollers und die Erstellung einer Messsoftware, welche das gesamte System steuert und Daten aufzeichnet. In verschiedenen Versuchsreihen müssen

dann aussagekräftige Messdaten ermittelt und ausgewertet werden. Die Arbeit gliedert sich also in vier Teile, hinzu kommen noch theoretische Erläuterungen zu den grundlegenden Prinzipien.

2 Theoretische Vorbetrachtungen

2.1 Modulsystem

Ausgangspunkt der Diplomarbeit ist das von der Lehr- und Forschungsgruppe Optronik entwickelte Modulsystem in der Version 5.1. Kernstück dessen ist der Mikrocontroller Atmega128. Dieser dient als Schnittstelle zwischen Messsoftware auf dem PC und der eigentlichen Messhardware, er übernimmt vom Steuerrechner Befehle und Anweisungen und gibt diese an die Hardware weiter. Ebenso nimmt er Daten von der Hardware auf und übergibt diese an den Host. Das Modulsystem verfügt in der aktuell verwendeten Version über eine Grundplatine mit Stromversorgung, Busleitungen und Steckplätzen, in die Module mit verschiedenen Funktionen und Aufgaben eingesetzt werden können. Das ermöglicht es, unterschiedliche Messmethoden und Elektronik zu benutzen und zu testen, ohne jedes Mal eine komplette Neuentwicklung zu machen. Die Prozessorplatine hat dabei eine Sonderstellung inne, da sie über zusätzliche Kontakte zum Anschluss der Grundplatine an ein Ethernet verfügt. Abbildung 2.1 zeigt das Modulsystem samt installierten Modulen.

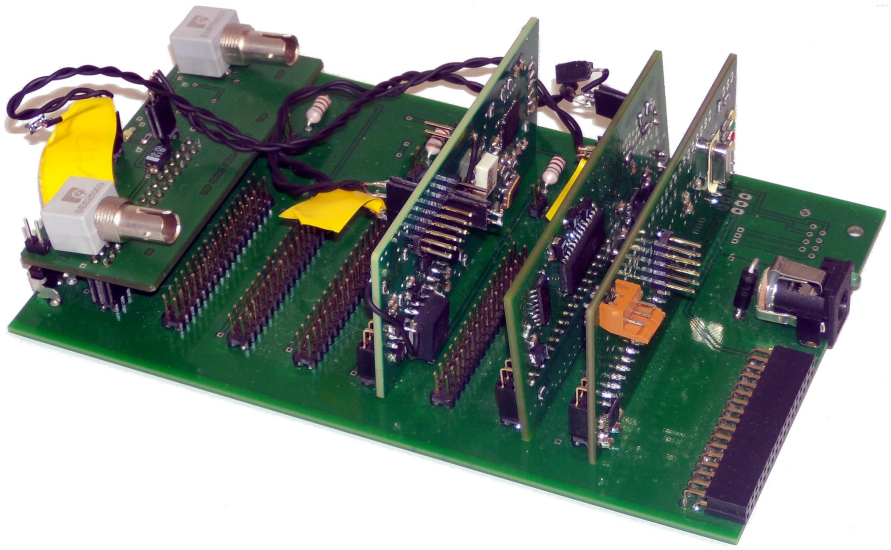

Abbildung 2.1: Modulsystem mit installierten Modulen

Für die Diplomarbeit wird in vielerlei Hinsicht auf bereits Bestehendes zurück gegriffen. Das betrifft die schon erwähnte Grundplatine, aber auch Module. Das Mikroprozessormodul, dass den ATmega128 und seine Außenbeschaltung aufnimmt, wird ebenso übernommen, wie das Sende- und Empfangsmodul, dass das generierte Messsignal auf einen Lichtträger für den optischen Wellenleiter aufmoduliert, sowie das Phasenkomparatormodul, das in Kapitel 3.2 noch einmal erläutert wird.

2.2 Integraler Längensensor

Ein integraler Längensensor, wie das Modulsystem es darstellt, dient zur Erfassung von Änderungen in der Länge des Sensormediums. Es wird dabei ein elektrisches oder optisches Signal mit einer bestimmten Frequenz in die Messstrecke eingekoppelt, welche als Schleife ausgeführt ist. Vergleicht man nun die Phase des Signals am Eingang mit der Phase des Signals am Ausgang der Schleife, so bilden sich Änderungen der Länge der Messstrecke als Veränderung der Phasenlage des Eingangs- zum Ausgangssignals ab. Abbildung 2.2 verdeutlicht das Grundschema. Ohne Regelung läuft das System als Open Loop, da man bei festen Eingangswerten nur beobachtet, wie sich der Ausgang verhält.

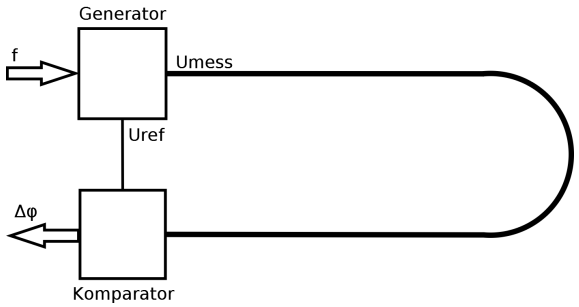

Abbildung 2.2: Längensensor als einfache Open Loop

Um aber eine stabile Messung zu ermöglichen und um Probleme mit der Periodizität der Phasenlage zu entgehen, wird das Eingangssignal in Frequenz oder Phase derart nachgeführt, dass sich am Faserausgang immer eine festgelegte Phasenlage einstellt. Man spricht man von einer Closed Loop. Aus der Nachführung des Eingangssignals lässt sich nun die Längenänderung über die Zeit ermitteln. Die Längeninformtation steckt im nachgeführten Parameter, womit sich die beiden folgend beschriebenen möglichen Trackingverfahren ergeben.

2.2.1 Frequenztracking

Nachgeregelt wird hierbei die Frequenz des eingekoppelten Signals. Es wird immer versucht die Phasenverschiebung zwischen Eingangs- und Ausgangssignal konstant zu halten, indem die Frequenz entsprechend erhöht oder verringert wird. Eine Verlängerung der Strecke führt zu einer Verringerung der Frequenz, denn der gedehnte Wellenleiter bildet am Ausgang einen „späteren“ Zeitpunkt der Welle ab als der ungedehnte.

Diese Trackingvariante ist mit vergleichsweise geringen Aufwand realisierbar, da lediglich ein frequenzmäßig fein einstellbarer und stabiler Generator benötigt wird. Abbildung 2.3 zeigt das Funktionsprinzip. Das Signal das Generators wird über einen Leistungsteiler aufgeteilt, um einmal das Referenz- (U_{ref}) und einmal das Messsignal (U_{mess}) für die Faser zur Verfügung zu stellen. Der Phasenkomparator vergleicht die Phasenlage der beiden Signale und gibt diese in einer Spannung kodiert an den Mikrocontroller weiter. Dieser überträgt die Daten an den Host, welcher die Messwerte sammelt und eine neue Frequenz

berechnet. Der Mikrocontroller stellt anschließend die Frequenz auf den neu errechneten Wert ein.

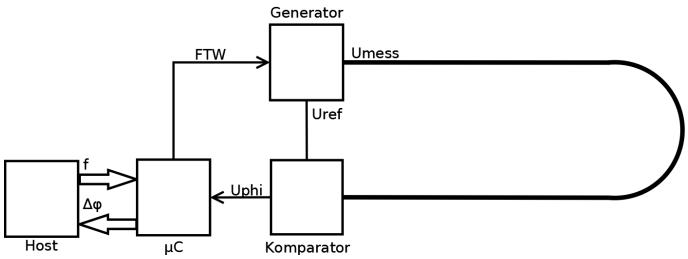

Abbildung 2.3: Längensensor mit f-Tracking

Problematisch ist jedoch, dass viele Bauelemente ein frequenzabhängiges Verhalten aufweisen. Ein Ändern der Frequenz, wenn auch nur geringfügig, kann die elektrischen Parameter der Bauteile im Signalpfad verändern. Insbesondere das Verhalten von optischen Wellenleitern variiert stark in Abhängigkeit von der Frequenz, aber auch die Sende- und Empfangsdioden zeigen ein frequenzabhängiges Verhalten.

2.2.2 Phasentracking

Wie Abbildung 2.4 zeigt, existieren beim Phasentracking zwei getrennt regelbare Generatoren, die ein Mess- und ein Referenzsignal liefern. Wird nun die Länge des Sensors verändert, wird die Phasenverschiebung am Messfasereingang zwischen den beiden Signalen nun entsprechend angepasst, indem die Phase des Messsignals variiert wird. Das Signal des Referenzgenerators wird direkt auf den Komparator gegeben. Beide Generatoren müssen absolut synchron zueinander laufen, um eine funktionierende Regelung zu ermöglichen.

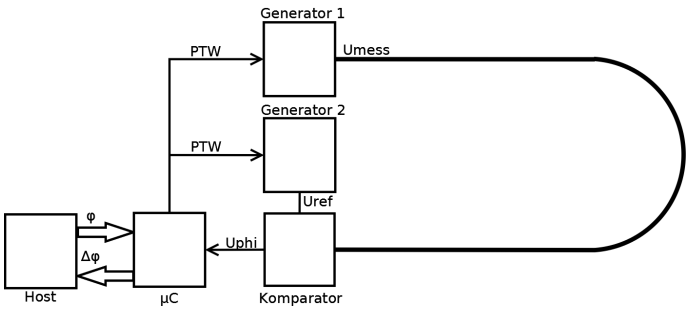

Abbildung 2.4: Längensensor mit ϕ-Tracking

Vorteilhaft hierbei ist nun, dass man eventuellen Frequenzabhängigkeiten aus dem Weg geht, da man bei einer festen Frequenz bleibt. Bauelemente- und Filterdimensionierung kann nun schmalbandiger ausfallen. Nachteilig sind hingegen die erhöhten Anforderungen an die Signalquellen bezüglich Synchronität und definierter Phasenverschiebung.

2.3 Direkte Digitale Synthese

Als Signalgeneratoren für Messsysteme, die ihre Informationen aus Phasenverschiebungen von Signalen ermitteln, eignen sich sogenannte DDS-Bausteine hervorragend. DDS steht für Direct Digital Synthesis (oder auch Direkte Digitale Synthese). DDS-Generatoren besitzen ein sehr geringes Phasenrauschen und sie erlauben es, fast beliebige Frequenzen mit hoher Genauigkeit zu erzeugen. Die Frequenz ist dabei nur durch die erforderlich Referenzfrequenz der DDS nach oben begrenzt. Bei einigen DDS-IC sind auch vielfache davon durch interne Multiplikation möglich, allerdings mit negativen Auswirkungen auf das Phasenrauschen.

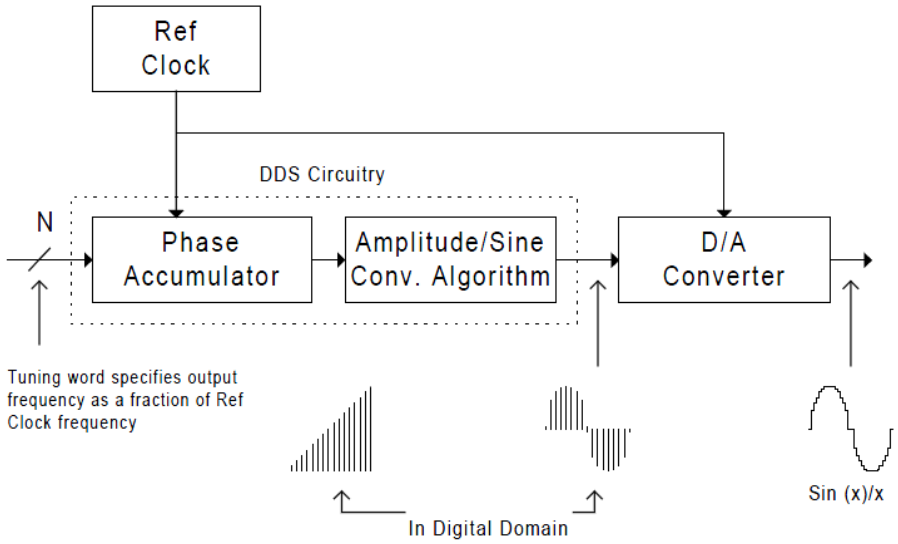

Abbildung 2.5: Prinzipieller Aufbau einer DDS[1]

Prinzipiell besteht eine einfache DDS, wie in Abbildung 2.5 ersichtlich, aus einem Referenztakt, einem Adresszähler, einem Speicher und einem DA-Wandler. Der Wert im Adresszähler wird im Takt der Referenz immer um eins erhöht. Wird der höchste Wert des Adresszählers erreicht, kommt es zu einem Überlauf und der Zähler zählt wieder ab null. Am Ausgang zeigt sich also eine diskret ansteigende Spannung in Form einer Sägezahnschwingung. Dieser Spannungswert stellt für den nachfolgenden Speicher, im einfachsten Fall ein PROM, eine Adresse dar. Der an dieser Adresse abgelegte Spannungswert wird ausgelesen und am Ausgang des Speichers bereit gestellt. Es lassen sich also verschiedene Spannungsverläufe ablegen, in aller Regel ist dies eine Sinusschwingung. Der diskrete Spannungsverlauf am Ausgang des Speichers wird anschließend noch durch einen Tiefpass geglättet.

Dieses einfache Grundprinzip erlaubt es allerdings nur, eine Schwingung mit einem festen Bruchteil der Referenzfrequenz zu erzeugen. Die Größe des Teilungsfaktors ergibt sich dabei aus der Wortbreite des Adresszählers. Um ein Variieren der Ausgangsfrequenz zu ermöglichen kommt zusätzlich ein Frequenzregister zum Einsatz, dass den bei jedem Takt der Referenz zum Adresszähler hinzuzuaddierenden Wert beinhaltet und überschrieben werden kann. So werden bestimmte Werte des Speichers übersprungen oder mehrfach

[1] Quelle: S.8 in [3]

ausgelesen, was Frequenzvariationen am Ausgang ermöglicht. Phasenänderungen sind durch hinzuaddieren oder -subtrahieren eines Offsets im Adresszähler möglich. Dadurch laufen die ausgelesenen Werte den eigentlich gezählten voraus oder hinterher, es kommt zu einem Phasensprung.

Dieses einfache Funktionsmodell ist das grundlegende Modell, nach dem reale DDS-Generatoren realisiert werden. Es stellt sich jedoch noch das Problem, dass große Speicher viel Chipfläche benötigen und Chipfläche wiederum sehr teuer ist. Um nun Kosten zu senken, wird die Symmetrie des Sinus ausgenutzt und nur das erste Viertel der Schwingung digital abgelegt und der Speicher so ausgelesen bzw. die Werte so manipuliert, dass sich wieder ein Sinus ergibt. Außerdem kommen in der Praxis noch, je nach DDS-Baustein, einige zusätzliche Funktionen und Register hinzu, die zum Beispiel ein schnelles Umschalten zwischen zwei festen Frequenzen oder Phasenlagen ermöglichen. Damit lassen sich zum Beispiel recht einfach digitale Daten auf hohe Frequenzträger aufmodulieren. Dies ist jedoch nicht Bestandteil der Diplomarbeit, da die DDS nur als fein einstellbare, stabile Frequenzquelle genutzt wird.

3 Praktischer Aufbau

Dieser Teil soll einen Überblick über technische Einzelheiten wichtiger verwendeter Bauteile und des Schaltungsaufbaus geben. Bei meiner Arbeit konnte ich auf einen sehr weit fortgeschrittenen Schaltungsentwurf für die Platine der Doppel-DDS von Herrn Dipl.-Ing. Mirko Mothes zurück greifen. An diesem waren ein paar Ergänzungen und Korrekturen vorzunehmen. Alle anderen Module lagen bereits vor und konnten unverändert genutzt werden.

3.1 AD9958

Der Doppel-DDS-Baustein AD9958 von Analog Devices verfügt über zwei unabhängig einstellbare Kanäle, welche beide in Frequenz und Phase variieren lassen. Da die DDS nur eine Referenzquelle nutzt, laufen beide Kanäle absolut synchron und lassen sich definiert zueinander in der Phase verschieben. Der genutzte Referenzquarz mit 156,25 MHz ist ausreichend größer getaktet als die später maximal genutzte Ausgangsfrequenz von 30 MHz, so dass das Ausgangssignal sehr störfrei erzeugt und auf den internen, PLL-basierten Frequenzmultiplikator verzichtet werden kann. Ein Großteil der Außenbeschaltung der DDS ist im Datenblatt[2] bereits vorgegeben.

3.1.1 Anschluss an den ATmega128

Die Kommunikation zwischen dem Mikrocontroller und der DDS erfolgt über SPI, ein einfaches Drei-Draht-Interface mit MISO (bei der DDS SDIO2), MOSI (bei der DDS SDIO0) und CLK. Die Interfacepins SDIO1 und SDIO3 werden für den SPI-Bus nicht benötigt und deshalb definiert mit 1 kOhm auf Masse gelegt.

Zusätzlich zu den drei SPI-Leitungen müssen noch die Anschlüsse IO_Update, CS und Reset zum Mikrocontroller geführt werden. IO_Update als auch Reset waren bei dem bisherigen Entwurf nicht zum Mikrocontroller geführt. Ein kurzer Impuls an IO_Update sorgt dafür, dass zur DDS übermittelte Anweisungen vom Eingangspuffer in die eigentlichen Register übergeben werden. Übertragene Anweisungen werden also erst mit diesem Signal aktiv. CS steht für Channel Select und dient der Adressierung mehrerer SPI-Geräte an einem Bus. Die Adressdecodierung erfolgt mit Hilfe des MC74AC138. Reset wird genutzt, um nach einem Neustart oder Reset des Modulsystems auch die DDS zurück zu setzen. Die

[2] siehe S.9f in [5]

Register der DDS sind nichtflüchtig und nach einem Wegfall und erneuter Inbetriebnahme der Spannungsversorgung arbeitet die DDS mit den zuletzt eingestellten Werten weiter.

3.1.2 Register

Die Einrichtung und Steuerung der Doppel-DDS erfolgt, wie schon erwähnt, über Register, die von Außen per SPI beschrieben werden müssen. Tabelle 3.1 listet alle verfügbaren Register auf und erklärt kurz deren Funktion. Jedes Register besitzt eine Adresse, die zunächst als Instruction Byte übermittelt werden muss. Mit einem Bit in diesem Byte wird festgelegt, ob dieses Register ausgelesen oder beschrieben werden soll. Details dazu finden sich auch Seite 32 in [5]. Anschließend müssen alle Bytes des Registers gelesen oder beschrieben werden. Dies wird von der DDS so erwartet und ein nicht einhalten stört den Komunikationsablauf mit der Folge, dass der Chip nicht mehr wie gewünscht gesteuert werden kann.

Die möglichen Einsatzzwecke der Doppel-DDS sind sehr vielfältig und komplex, was sich an der Vielzahl an Registern und deren Funktionen widerspiegelt. So lässt sich dank Speicher für mehrere Frequenz-, Amplituden oder Phasenworte mit zusätzlichen Pins schnell zwischen verschiedenen Frequenzen, Phasen oder Amplituden umschalten, um zum Beispiel eine FSK-, PSK- oder ASK-Modulation durchzuführen. Möglich sind auch Sweeps für einen der Parameter. Für diese Diplomarbeit sind jedoch nur wenige Register relevant, da die grundlegende Möglichkeit der Frequenz- und Phaseneinstellung genügt.

Name	Bits	Funktion
Channel Select Register (CSR)	8	Kanalauswahl und Schnittstellenkonfiguration
Function Register 1 (FR1)	24	diverse Konfigurationsbits, Einstellen der internen PLL
Function Register 2 (FR2)	16	diverse Konfigurationsbits
Channel Function Register (CFR)	24	diverse Kanaleinstllungen, u.a. Skalierung des Ausgangsstroms
Channel Frequency Tuning Word (CFTW)	32	zu nutzendes Frequenzwort
Channel Phase Offset Word (CPOW)	16	Phasenoffset
Amplitude Control Register (ACR)	24	Amplitudensteuerung
Linear Sweep Ramp Rate (LSRR)	16	Verweilzeit linearer Anstieg/Abfall
LSR Rising Delta Word (RDW)	32	Schrittweite linearer Anstieg
LSR Falling Delta Word (FDW)	32	Schrittweite linearer Abfall
Channel Word 1-15 (CW)	32	zur Nutzung von Frequenz-, Phasen- oder Amplitudenmodulation

Tabelle 3.1: Register des AD9958

Als erstes muss das Channel Select Register gesetzt werden, da dieses Informationen darüber enthält, wie die SDIO-Pins genutzt werden und ob das LSB oder das MSB zuerst übertragen wird. Das Interface lässt sich zwischen Two/Three Wire Serial Mode, 2-Bit Serial Mode und 4-Bit Serial Mode umschalten. Im Modulsystem wird der Three Wire Serial Mode genutzt, da er mit dem SPI des ATmega128 identisch ist. Des weiteren erfolgt über dieses Register die Kanalauswahl. Da sämtliche Register bis auf CSR, FR1 und FR2 für jeden Kanal einzeln vorhanden sind, sie im Eingangspuffer aber nur einmal existieren, muss mit den Channel Select Bits eine Kanalauswahl getroffen werden, damit die Einstellungen bei einem I/O_Update an der richtigen Stelle landen.

Beide Function Register bieten viele Konfigurationsbits. Die Funktionen werden aber nicht benötigt und deshalb null gesetzt, somit also deaktiviert. In den Channel Function Register muss für beide Kanäle der DAC-Ausgangsstrom mit Bit *8* und *9* auf vollen Ausgangsstrom gesetzt werden.

Die Register Channel Frequency Tuning Word und Channel Phase Offset Word dienen zum Einstellen der Frequenz und der Phaselage der beiden Kanäle. Natürlich sind diese Register für jeden Kanal getrennt vorhanden, was eine unabhängige Steuerung erlaubt. Das Frequency Tuning Word (FTW) ist 32 Bit lang und berechnet sich nach Formel 3.1, wobei m für die Auflösung des Frequenzwortes steht, f_{ref} für den Takt des Referenzquarzes und f_{out} für die einzustellende Frequenz[3]. Dagegen besitzt das Phase Tuning Word (PTW) 16 Bit, wovon aber nur 14 Bit Auflösung von der DDS verarbeitet werden. Die Berechnung erfolgt analog zum FTW nach Formel 3.2 mit n für die Auflösung des Phasenwortes und ϕ_{out} für den gewünschten Phasenoffset[4]. Dabei ergibt sich natürlich die Phasenlage beider Kanäle zueinander durch die Offsets beider Kanäle, weshalb im weiteren Verlauf ein Kanal immer mit einem Offest von 0° beaufschlagt wird. Die genauen Konfigurationsdaten sind auch im Quelltext des Mikrocontrollerprogramms im Anhang auf Seite 41ff in der DDS-Init-Routine zu finden.

$$FTW = \frac{2^m}{f_{ref}} * f_{out} \tag{3.1}$$

$$PTW = \frac{2^n}{360°} * \phi_{out} \tag{3.2}$$

3.1.3 Pegelanpassung

Problematisch ist die direkte Verbindung zwischen Mikrocontroller und DDS. Der ATmega128 arbeitet an seinen I/O-Pins mit Pegeln von 0 V bis 5 V, der AD9958 jedoch mit 0 V bis 3,3 V. Dies erfordert den Einsatz von Pegelwandlern, in dem Fall der SN74LVC2T von Texas Instruments. Dieser Baustein ist in der Lage den Pegel zweier Signale zu Wandeln, und das mit wählbarer Signalrichtung. So kommt ein Wandler für Channel Select und IO_Update und einer für MOSI und CLK zum Einsatz.

[3] Basierend auf der Formel auf S.8 in [3]
[4] Basierend auf der Formel auf S.18 in [5]

Ursprünglich war vorgesehen, dass auch MISO, der Rückkanal von der DDS, über einen Wandler zum Mikrocontroller geführt wird. Jedoch ist der Wandler nicht ganz unproblematisch zu handhaben. Der zweite Kanal dieses Wandlers sollte ursprünglich, weil nicht benötigt, auf Masse gelegt werden. Durch die interne Beschaltung führt dies jedoch dazu, dass auch der MISO-Kanal auf Masse gezogen wird, was wiederum den gesamten SPI-Bus erheblich stört und eine erfolgreiche Programmierung der DDS verhindert. Hinzu kommt, dass das erfolgreiche Auslesen der DDS-Register von Analog Devices nicht garantiert wird[5] und für die Anwendung auch nicht notwendig ist. Deshalb wird auf den Rückkanal derzeit einfach verzichtet und der dritte Wandler nicht verbaut. Siehe dazu auch die Anmerkung im Schaltplan.

3.1.4 Dimensionierung des Ausgangsfilters

Um einen möglichst störungsfreien, sinusförmigen Spannungsverlauf am Ausgang der DDS zu erhalten, muss zunächst das Ausgangssignal über einen eins-zu-eins-Transformator gewandelt werden. Der DAC des AD9958 arbeitet als Stromquelle und sollte nicht überlastet werden, da es bei zu hohen Ausgangsspannungen zu Verzerrungen kommt. Ein Transformator gewährleistet den definierten Abschluss des Ausgangs.

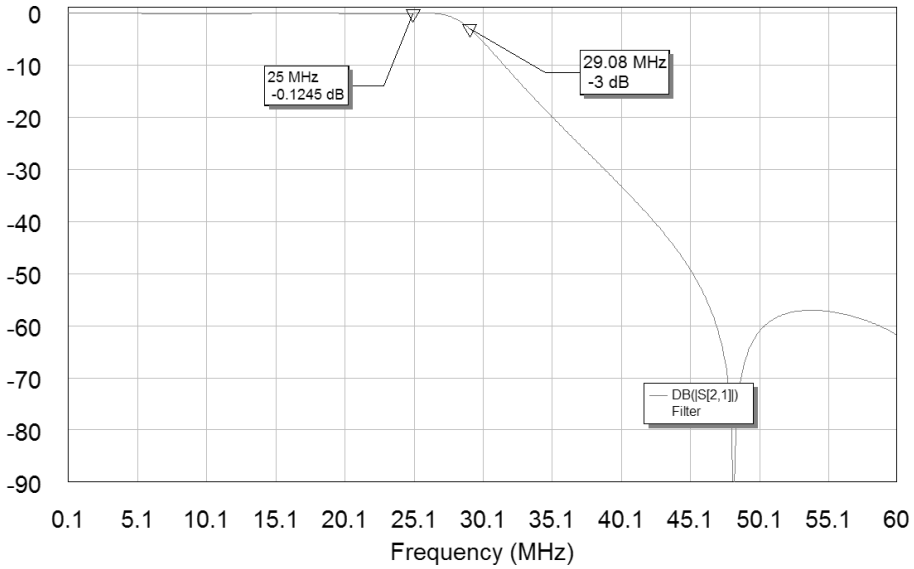

Abbildung 3.1: Übertragungsfunktion des Ausgangsfilters

Anschließend muss das Signal noch gefiltert werden, um technisch bedingt entstehende Oberwellen zu dämpfen. Als grundlegende Filtercharakteristik wird das Cauer-Filter gewählt, da dies bei minimalen Schaltungsaufwand einen sehr steilen Übergang vom Durchlass- zum Sperrbereich ermöglicht[6]. Der Entwurf erfolgt mit Hilfe der Software Ansoft Designer, die Simulation und Anpassung der Bauteilwerte an reale Größen mit Hilfe von AWR Microwave Office. Das Filter wird so angelegt, dass die Grenzfrequenz in etwa bei 30 MHz liegt. Der letztendlich in der Simulation erreichte Frequenzverlauf ist in Abbildung

[5] siehe dazu die Frage „Can I read back data at the same rate that I can write the data to the DDS device?“ unter [1]

[6] siehe hierzu auch den Wikipedia-Artikel unter [2]

3.1 zu sehen, der Aufbau und die Dimensionierung des Filters ist im Schaltplan (Abbildung A.8 im Anhang auf Seite 38) rechts unten zu finden.

3.1.5 Gesamter Platinenaufbau

Der komplette Schaltplan findet sich im Anhang auf Seite 38f, das erstellte Layout auf Seite 40. Neben der DDS mit seinem Referenzquarz, den Pegelwandlern und dem Ausgangsfilter befinden sich noch Spannungswandler auf dem Modul, die die notwendigen 1,8 V Betriebsspannung und 3,3 V I/O-Spannung bereit stellen. Diese sind notwendig, da das Modulsystem nur 5 V und 9 V zur Verfügung stellt. Die notwendige Adressdecodierung erfolgt über den Demultiplexer MC74AC138, der auch auf anderen Modulen zum Einsatz kommt.

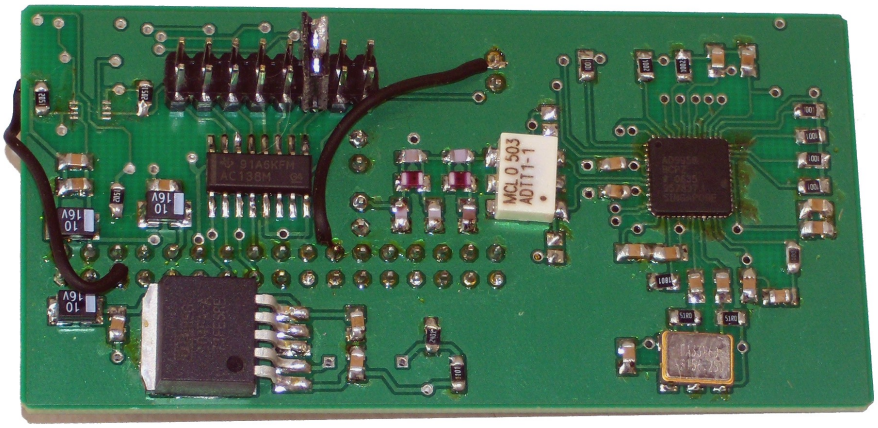

Abbildung 3.2: Fertig bestücktes Modul, Vorderseite

Abbildung 3.2 zeigt das fertig bestückte Modul von der Vorderseite. Rechts ist die DDS samt Referenzquarz, mittig der Übertrager und das Ausgangsfilter vom ersten Kanal, links oben der Adressdecoder und links unten der Spannungswandler für 1,8 V. Zu sehen sind außerdem noch zwei nachträglich eingefügte Drahtbrücken, die unbedingt notwendig sind. Die linke ist der Anschluss des MOSI-Kanals an den Pegelwandler. Diese Verbindung ist einer späteren Schaltplanoptimierung zum Opfer gefallen. Die rechte ist der Anschluss des µC an den Reset-Pin der DDS, dessen Notwendigkeit sich erst bei der Inbetriebnahme der DDS zeigte. Die Fehler in den Dokumenten wurde aber behoben. Die Rückseite (Abbildung 3.3) beherbergt links oben den Spannungswandler für die 3,3 V, mittig den Übertrager und das Ausgangsfilter für den zweiten Kanal sowie rechts daneben die Anschlusspins für die Kanäle, die Anschlussleiste und rechts oben die zwei verwendeten Pegelwandler.

3.2 Der Phasendetektor

Die Phasendetektorplatine ist ein bereits vorhandenes Modul. Für die Diplomarbeit ist jedoch eine genaue Kenntnis des verwendeten Phasendetektor AD8302 von Analog Devices

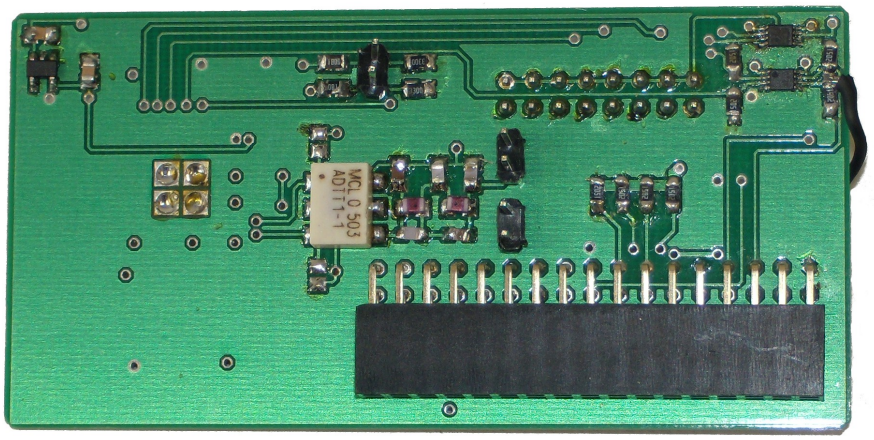

Abbildung 3.3: Fertig bestücktes Modul, Rückseite

erforderlich. Dieser besitzt zwei Eingänge und ist in der Lage, Amplituden- und Phasenunterschiede zwischen zwei anliegenden Signalen zu detektieren. An ihm wird zum einen das Signal von Kanal 1 des AD9958 direkt als Referenz angeschlossen, zum anderen das Ausgangssignal der optischen Faser als Messsignal. Der Detektor gibt an seinen Ausgängen zwei Gleichspannungen aus, VMAG und VPHS. VMAG besitzt eine weitgehend lineare Abhängigkeit vom logarithmierten Amplitudenverhältnis der beiden Eingangssignale mit einem Anstieg von 30 mV/dB. VPHS ist linear abhängig zum Betrag der Phasenverschiebung der Eingangssignale mit einem Anstieg von 10 mV/Grad. In Abbildung 3.4 sind die Ausgangsspannungen im Verhältnis zu den Eingangsbedingungen dargestellt.

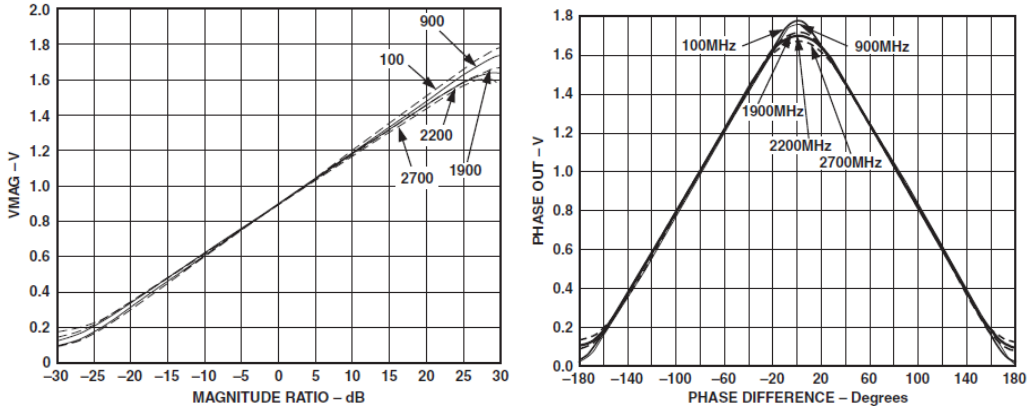

Abbildung 3.4: Verlauf von VMAG und VPHS[7]

VMAG und VPHS werden mit Hilfe eines AD974, ein 4-Kanal AD-Wandler von Analog Devices mit 16 Bit Auflösung, digitalisiert. VMAG wird mitprotokolliert, um Dämpfungsänderungen durch äußere Einflüsse festzustellen und sicher zu gehen, dass an beiden Eingängen des AD8302 noch für eine Verwertung ausreichend hohe Pegel anliegen. Deren Empfindlichkeit ist natürlich nicht beliebig groß. Ebenso wird die Temperatur mittels eines einfachen Temperatursensors erfasst, welcher sich in der Nähe des AD8302 befindet. Dies dient dazu, eventuelle Temperaturabhängigkeiten beim Messen festzustellen und bei Bedarf

[7] Quelle: S.6 und S.10 in [4]

aus den Messwerten rauszurechnen. Die Ergebnisse des Phasendetektors sind auch recht strak temperaturabhängig. Der AD974 ist über SPI mit dem Mikrocontroller verbunden. Er startet die Digitalisierung, sobald das entsprechende Signal vom Controller gesetzt wird und signalisiert an diesem wiederum das Ende der Wandlung. Die Daten können dann gelesen werden. Der AD8302 läuft als analoger Schaltkreis hingegen kontinuierlich und stellt dauerhaft seine Ergebnisse am Ausgang zur Verfügung.

4 Softwarelösungen

Ein wichtiger und umfassender Bestandteil der Arbeit ist die Programmierung sowohl des Mikrocontrollers des Messsystems als auch der Messsoftware des Hostrechners. Erst der µC ermöglicht die Steuerung der Doppel-DDS sowie des AD-Wandlers. Er dient als Komunikationsschnittstelle zwischen der Hostsoftware und der Messhardware und muss über die serielle Schnittstelle mit dem Hostsystem kommunizieren.

4.1 Mikrocontrollerprogrammierung

Die Programmierung des µC erfolgt in C. Als Entwicklungsumgebung dient AVR-Studio mit WinAVR als Compiler. Einige Routinen können komplett oder zumindest mit etwas Anpassung vom bisherigen Sensorsystem und seinem µC-Programm übernommen werden. Dies spart zum einen Zeit, zum anderen verringert es potentielle Fehlerquellen, da die Software bereits mehrfach erfolgreich verwendet wurde. Im Quelltext im Anhang auf Seite 41ff wurde versucht kenntlich zu machen, was die Eigenleistungen dabei sind.

4.1.1 SPI

Der SPI-Bus ist ein einfaches Bussystem zur seriellen, synchronen, bidirektionalen Übertragung von Daten. Dabei ist immer ein Teilnehmer am Bus der Master, die restlichen Geräte sind Slaves. Eine Übertragung geht immer vom Master aus. Im Bereich der µC ist dieses Bussystem weit verbreitet, da es auf einfache Weise den Anschluss mehrerer Peripheriegeräte ermöglicht. Dafür existieren zwei Datenleitungen, eine liefert Daten vom Master zum Slave, die andere Daten vom Slave zum Master. Der Synchrontakt wird über die dritte Leitung vom Master ausgegeben. Bei mehereren Geräten ist auch eine Adressdecodierung mit einer Channel-Select-Leitung notwendig. Die Codierung der zu übertragenden Daten ist jedoch von den Geräten abhängig, da eine Vielzahl von unterschiedlichen Geräten mit verschiedensten Funktionen den SPI-Bus nutzen. Je nach Gerät sind auch noch zusätzliche Leitungen erforderlich. Konkret bei der DDS die besagten IO_Update und Reset, beim AD974 Steuerleitungen zum Starten des Wandlers und als Rückkopplung sowie Kanalwahlleitungen.

Die Konfiguration des ATmega128 erfolgt gemäß den Application Notes[8]. Da der µC bereits über ein in die Hardware integriertes SPI-Interface verfügt, ist es lediglich notwendig die Konfigurationsregister korrekt einzustellen. Die Einstellungen sind identisch mit der

[8] siehe Kapitel „Serial Peripheal Interface - SPI“ auf S. 163ff in [6]

Konfiguration aus früheren Implementierungen der Lehr- und Forschungsgruppe. Angepasst werden muss nur das Registerbit CPHA (Clock Phase). Es ist auf „1“ zu setzen, damit die Daten auf dem SPI-Bus mit der steigenden Flanke des Takts gültig sind und auch gelesen werden. Der AD9958 setzt dieses Busverhalten voraus[9], während der AD974 auch mit gültigen Werten bei fallender Flanke umgehen kann.

Um Daten an die Doppel-DDS zu Übertragen wird die Funktion SPI_2DDSSend genutzt. Diese erwartet als Argumente immer die Registeradresse (siehe Abschnitt 3.1.2), die Anzahle der zu übertragenden Bytes sowie die Daten. Damit wird sicher gestellt, dass die Register immer korrekt beschrieben werden. Im Program werden diese Zugriffe noch weiter gekapselt, also nur noch von anderen Funktionen aufgerufen, um Fehler beim Beschreiben zu verhindern. Zur Übertragung vom Frequenzwort dient die Routine SET_FREQUENCY, für die Phase SET_PHASE. Beide beginnen damit, über die Funktion SPI_ADR die Adresspins des ATmega128 so zu schalten, dass der Channel-Select der DDS aktiviert wird. Dann wird mit die Funktion CHANNEL_SELECT zunächst der passende Befehl für die Kanalauswahl an das Channel-Select-Register der DDS über SPI geschickt. Anschließend wird das Frequenz- oder Phasenwort an das passende Register mit der Funktion SPI_2DDSSend geschickt. Zu Letzt wird ein IO_Update ausgelöst, damit die Einstellungen auch aktiv werden, und das Chip-Select wieder deaktiviert. Werden nun nur die oberen Funktionen SET_FREQUENCY und SET_PHASE mit ihren Parametern channel und FTW bzw. PTW benutzt, braucht man sich um die korrekte Kommunikation mit der DDS nicht mehr kümmern und kann so Fehler vermeiden.

4.1.2 USART

Der USART ist eine einfache und vielseitige serielle Kommunikationsschnittstelle, die zum Anschluss des Messsystems an einen Hostrechner über die RS232-Schnittstelle genutzt wird. Da sie ebenfalls bereits in der Hardware des ATmega128 integriert ist, ist die Verwendung ohne großen Aufwand möglich. Die Routine zur Konfiguration und zur Interruptbehandlung des USART wird aus bereits vorhandenen Implementierungen übernommen. Dies vereinfacht die Programmierung, da eine eventuelle Fehlerquelle vermieden wird.

Angepasst wurde dabei die Interruptroutine SIG_USART0_RECV, die Daten vom Interface entgegen nimmt, sobald neue anliegen. Da für die Doppel-DDS neben dem Auswahlbyte auch noch das Steuerbyte und Daten von bis zu 4 Byte Länge übertragen werden müssen, muss der Interrupt diese Daten auch korrekt annehmen und ablegen. Dies erfolgt zunächst mit Hilfe eines Arrays. Anschließend wird in der Hauptroutine das nachfolgend beschriebene Befehlsbyte der DDS analysiert, die Daten zu Kanalauswahl und Frequenz- oder Phasenwort in Variablen abgelegt und die entsprechenden Funktionen aufgerufen. Dies ist notwendig, da ein Zerlegen des Steuerwortes in der Interruptroutine womöglich zu lange dauern würde und Daten bei der Übertragung verloren gehen könnten. Die Funktionen, die Anweisungen für den AD974 bearbeiten und seine Wandlungsergebnisse an den Host senden wurden komplett übernommen.

[9] siehe hierzu in [5] S.31 und in [6] S.170

4.1.3 DDS-Befehle

Die Kommunikation zwischen Mess-PC und Mikrocontroller über RS232 erfolgt mit einfachen ASCII-codierten Befehlen. Dabei werden verschiedene Geräte am SPI-Bus über eine zugewiesene Zahl ausgewählt und anschließend ein oder mehrere Befehlswörter für dieses Gerät übertragen. Dies ist notwendig, um Setzen einer neuen Frequenz in der DDS durch die Hostsoftware oder ein Auslesen des AD-Wandlers zu ermöglichen.

Die Doppel-DDS bekommt in diesem Zusammenhang den Code „5“ zugewiesen, welcher sich hexadezimal als 0x35 ausdrückt. Dieser Code ist von bisherigen Modulen noch nicht belegt worden. Da die DDS eine im Vergleich zu den sonstigen Bauelementen größere Befehlspalette benötigt, um voll genutzt werden zu können, wurde ein einfaches Protokoll für Datenübertragung von Doppel-DDS-Anweisungen festgelegt, welches in Tabelle 4.1 zu sehen ist.

0	1	2	3	4	5
Auswahl der 2DDS 0x35h	Steuerbyte	Befehlsbytes			

Tabelle 4.1: Protokoll für die Doppel-DDS

Nach Übertragung des Auswahlbytes zur Selektion der Doppel-DDS folgt ein Steuerbyte, in welchem mitgeteilt wird, ob die Frequenz oder die Phase mit den angehangenen Daten gesetzt werden soll, und welche Kanäle betroffen sind. In Tabelle 4.2 ist die dabei zu Grunde liegende Codierung dargestellt. Bit *S3* wählt das Phasenregister, Bit *S2* das Frequenzregister. Aus naheliegenden Gründen ist nicht möglich, Frequenz und Phase gleichzeitig zu setzen. Dies wird überprüft und löst bei Bedarf eine Übertragungsfehlermeldung aus. Die Bits *S1* und *S0* erlauben die Wahl der Kanäle nach Tabelle 4.3.

S7	S6	S5	S4	S3	S2	S1	S0
x				Ph	Freq	CH_Sel	

Tabelle 4.2: Bitcodierung Steuerbyte Doppel-DDS

S1	S0	
0	0	kein Kanal
0	1	Kanal 0
1	0	Kanal 1
1	1	beide Kanäle

Tabelle 4.3: Kanalauswahl über CH_Sel

Abschließend folgen 4 Byte mit dem zu setzenden Wert für die Doppel-DDS. Insgesamt ist also jeder DDS-Befehl 6 Byte lang. Man könnte den Befehl für ein Phasenwort auch um zwei Byte verkürzen, da die sich die Phase nur mit einer Auflösung von 14 Bit steuern lässt. Dies würde jedoch eine etwas aufwändigere Auswertung in der USART-Interrupt-Routine erfordern, welche zu einem größeren Zeitbedarf führen würde und somit die Gefahr mit sich bringt, dass Daten bei der Übertragung verloren gehen, da der Eingangspuffer des USART nicht rechtzeitig geleert werden kann.

4.2 Hostsoftware

Die Steuer- und Messsoftware des Hostrechners wird mit Hilfe von Visual Studio 2008 in VB.net programmiert. Das Programm erledigt die Kommunikation über RS232 mit dem µC, die Regelung nach dem Frequenz- oder Phasentrackingverfahren, Messwerterfassung, -darstellung und -aufbereitung. Die Bedienung der Oberfläche sowie technische Programmierdetails sollen folgend erläutert werden.

4.2.1 Regelung

Grundlage für den Regelalgorithmus ist die Profilmessung der verwendeten Messfaser. Bei diesem erfolgt ein Frequenz- beziehungsweise ein Phasensweep über einen frei wählbaren Bereich. Die sich ergebenden Verläufe für Phasendifferenz, enthalten in VPHS, und Dämpfung, enthalten in VMAG, werden über den eingestellten Sweepbereich aufgetragen. In diesem Profil kann nun ein gewünschter Arbeitspunkt, bestehend aus Startfrequenz bzw. -phase und VPHS, ausgewählt werden. Idealerweise liegt dieser in einem möglichst linearen Bereich des Profils und nahe 90° Phasenverschiebung, entsprechend 900 mV VPHS. Dies sorgt für eine fehlerfreie und stabile Regelung, da man so den größten Abstand zu mehrdeutigen Messergebnissen ermöglicht und die weitere Regelung auf der Annahme eines weitgehend linearen Verlauf beruht.

Die Regelkonstante wird nun aus dem Profil rechnerisch ermittelt. Dazu wird ein Wertebereich festgelegt, welcher den gewählten Arbeitspunkt einschließen sollte. Aus den Profildaten im eingestellten Bereich wird nun mittels linearer Regression der Anstieg des Profils, die Regelkonstante, berechnet. Die Subroutine btnCalcK_Click() ist dafür zuständig. Im folgenden werden die genauen Regelalgorithmen für eigentlichen Messungen erläutert.

Frequenztracking

Die Regelung beim Frequenztracking basiert auf Formel 4.1. Bei jedem Aufruf der Regelfunktion wird zunächst die Spannung U_ϕ, identisch mit VPHS, vom AD8302 ausgelesen. Diese beinhaltet die Phasenverschiebung zwischen Referenz- und Messsignal. U_{AP} steht für den bei Beginn der Messung gewählten Arbeitspunkt der Spannung VPHS. Die Differenz der beiden Spannungen wird durch den Anstieg K der Profilkurve dividiert und das Ergebnis auf den beim letzten Aufruf eingestellten Frequenzwert aufaddiert.

$$f_{neu} = f_{alt} + \frac{U_{AP} - U_\phi}{K} \tag{4.1}$$

Phasentracking

Beim Phasentracking läuft die Regelung analog zu der beim Phasentracking ab. U_ϕ wird ausgelesen, von der Spannung des gewählten Arbeitspunkts abgezogen und das Ganze durch den Anstieg K dividiert. Dieser Differenzwert wird nun ebenfalls auf die zuletzt eingestellte

Phasenverschiebung aufaddiert. Der Anstieg K besitzt bei beiden Trackingvarianten auf Grund der vorherigen Profilmessungen die korrekten Einheiten.

$$\phi_{neu} = \phi_{alt} + \frac{U_{AP} - U_{\phi}}{K} \tag{4.2}$$

Beispielhafte Herleitung der Formel

Im Folgenden soll an Hand des Frequenztracking erläutert werden, wie die Formel für den Regelalgorithmus zu Stande kommt. In Abbildung 4.1 ist schematisch das Profil einer Messfaser über der Frequenz als durchgehende Kurve aufgetragen. Der zu Beginn der Messung gewählte Arbeitspunkt ist als Punkt AP samt zugehöriger Frequenz f_{AP} und Spannung U_{AP} eingezeichnet.

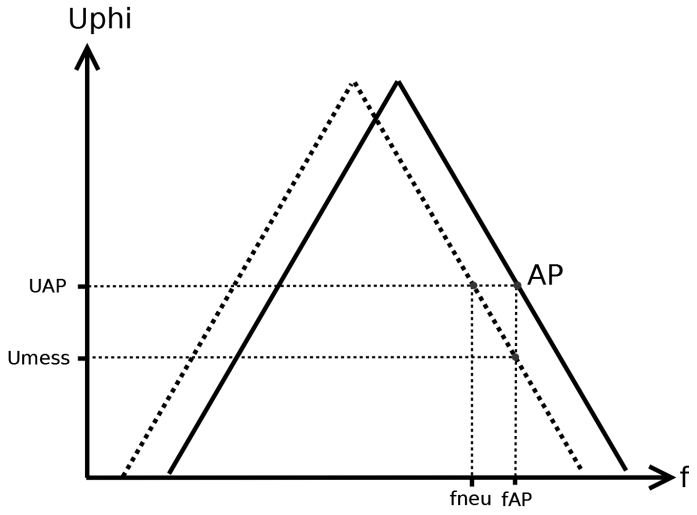

Abbildung 4.1: Herleitung der Formel zum Frequenztracking

Kommt es nun zu einer Dehnung der Faser, verschiebt sich der Verlauf des Profils so, wie es die gestrichelte Kurve darstellt. In Folge dessen wird bei gleicher Frequenz f_{AP} eine niedrigere Spannung U_{mess} gemessen. Ziel des Regelalgorithmus ist es aber, die Spannung und damit die Phasenverschiebung zwischen Ein- und Ausgang der Faser konstant zu halten. Dafür ist es nun notwendig, dass die Frequenz herab gesetzt wird.

$$\Delta f = \frac{\Delta U}{K} \tag{4.3}$$

$$\Delta U = U_{AP} - U_{mess} \tag{4.4}$$

$$\Delta f = f_{neu} - f_{alt} \tag{4.5}$$

Δf ergibt sich nach Formel 4.3 als Quotient aus Spannungsdifferenz und Anstieg K des Profils. Für die Spannungsdifferenz gilt die Festlegung nach Formel 4.4. Da für Δf ebenfalls

auch Formel 4.5 gilt, ergibt sich durch Einsetzen von Formel 4.3 in Formel 4.5 und Umstellen nach f_{neu} die Formel des Regelalgorithmus 4.1.

4.2.2 Visualisierung und Aufbau

Die grafische Benutzeroberfläche ist hauptsächlich auf Funktion als auf Benutzerfreundlichkeit ausgelegt. So fehlen einige Mechanismen um Fehler, insbesondere Eingabefehler, abzufangen oder den Ablauf der einzelnen Schritte zu kontrollieren. Ein korrekte und aussagekräftige Messung hängt also mit der Kenntnis und der korrekten Bedienung des Programms zusammen. Es folgt daher eine kleine Beschreibung der Benutzeroberfläche und dem allgemeinen Messablauf.

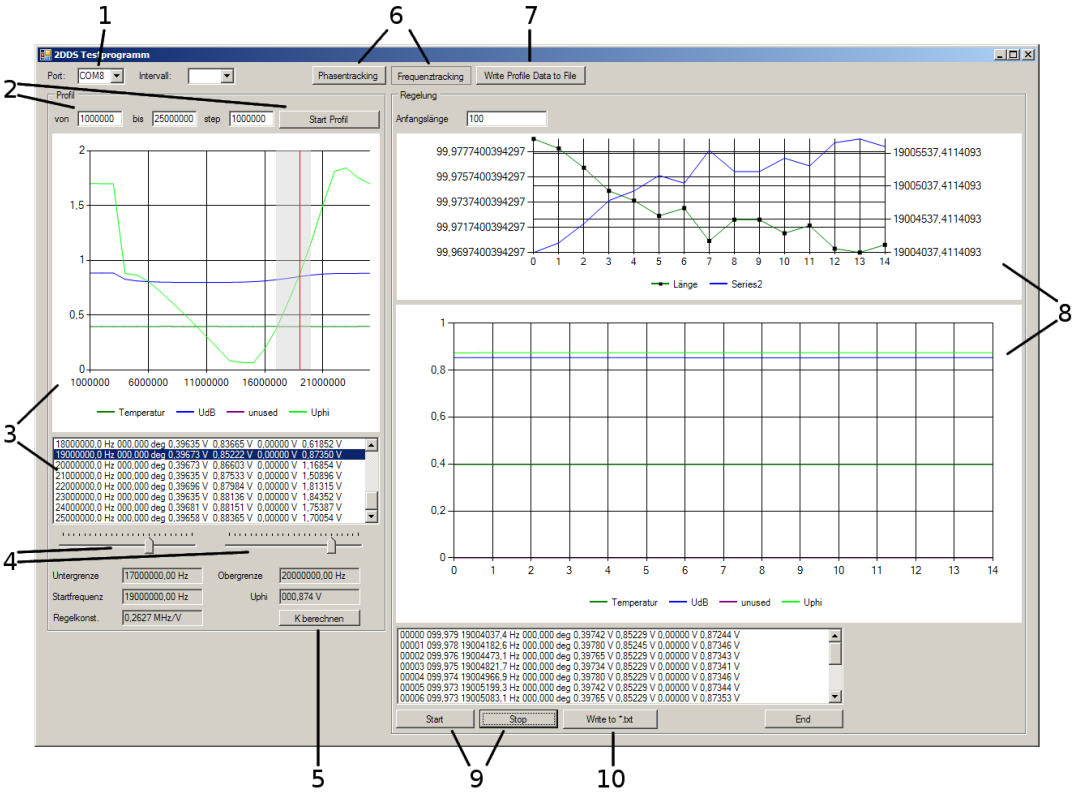

Abbildung 4.2: GUI der Hostsoftware

Abbildung 4.2 zeigt exemplarisch einen Screenshot der Hostsoftware nach Profil- und Längenmessung. Zunächst muss über *1* die verwendete COM-Schnittstelle, an welcher das Modulsystem angeschlossen ist, ausgewählt werden. Da diese Einstellung in der XML-Datei des Programms abgelegt und beim Start geladen wird, kann es zu Abstürzen beim Programmstart kommen, wenn der abgelegte COM-Port nicht auf dem Host vorhanden ist. In diesem Fall muss die XML-Datei manuell in einem Editor angepasst werden. Über die Buttons bei *6* lässt sich das Programm zwischen Frequenz- und Phasentracking umschalten. Dies sollte nicht zwischen Profil- und Längenmessung erfolgen, da dies zu völlig unbrauchbaren Daten führen würde.

Jede Messung beginnt, wie bereits erwähnt, grundsätzlich mit einer Messung des Profils. Dazu wird in den Feldern unter *2* die Start-, Stopfrequenz sowie Schrittweite in Hz

bzw. Anfangs- und Endphasenverschiebung sowie Schrittweite in deg angegeben und mit dem Button die Profilmessung gestartet. Unter *3* werden oben die Spannungen des AD8302 in Abhängigkeit von aktueller Frequenz bzw. Phase grafisch dargestellt und darunter noch einmal numerisch aufgelistet. Nach Abschluss der Profilmessung lässt sich der gewünschte Arbeitspunkt in der Liste auswählen. Der Arbeitspunkt für Frequenz bzw. Phasenverschiebung werden ebenso wie U_{AP} automatisch übernommen und auch in der Profilgrafik angezeigt. Die Regler bei *4* dienen der Festlegung der Grenzen zur Berechnung des Anstiegs K. Der gewählte Bereich wird ebenfalls in der Profilgrafik dargestellt und sollte weitgehend linear sein und den gewählten Arbeitspunkt einschließen, um einen stabilen Regelbetrieb zu gewährleisten. Mit Button *5* wird die Berechnung von K ausgelöst und mit Button *7* können die Profilmesswerte abgespeichert werden. Die eigentliche Messung kann beginnen.

Diese wird mit dem Start-Button unter *9* gestartet und kann mit dem Stop-Button beendet werden. Man beachte, dass ein pausieren nicht implementiert ist. Das Starten löscht alle bisherigen Messwerte. Die beiden Chart-Bereiche unter *8* stellen die Messwerte grafisch dar. Im oberen Feld wird versucht, die gemessenen Spannungswerte in Längen zurück zu rechnen und grafisch darzustellen, was aber noch nicht ausgreift und für den weiteren Verlauf der Diplomarbeit nicht relevant ist. Die dort dargestellten Daten existieren nur aus Testgründen. In der Fläche darunter werden wieder die Spannungswerte über die Messpunkte und damit über den zeitlichen Verlauf dargestellt. Diese Daten dienen der Diplomarbeit dann als Bewertungsgrundlage über die Vor- und Nachteile der Trackingverfahren. Nach Beenden einer Messung lassen sich die Messwerte mit Button *10* abspeichern.

4.2.3 Speichern der Messwerte

Messungen sowie Profile können mit einem einfachen Klick abgespeichert werden. Die Wahl des Speicherortes sowie des Dateinamens erfolgt automatisch. Im Verzeichnis der EXE-Datei wird, je nach Bedarf, ein Verzeichnis namens 'Profile' und ein Verzeichnis namens 'Messungen' erstellt. Ersteres enthält die Messwerte der Profile, zweiteres die Messwerte der eigentlichen Messungen. Beide Verzeichnisse enthalten jeweils, nach Bedarf, bis zu zwei Unterverzeichnisse, 'Frequency', (enthält Messwertdateien des Frequenztrackings) und 'Phase' (enthält Messwertdateien des Phasentrackings). Es ist daher notwendig, dass das Programm in einem Ordner ausgeführt wird, in dem der Nutzer Schreibrechte unter Windows besitzt.

Die Dateinamen werden aus dem Datum und der Uhrzeit zum Speicherzeitpunkt erstellt. Die Form ist dabei 'YYYY.MM.DD - HHmmss.txt', also Jahr, Monat, Tag, Stunden, Minuten und Sekunden. Damit wird verhindert, dass Namen doppelt auftauchen und eventuell vorhandene Messungen überschrieben werden. Die Datumsangabe beginnend mit Jahr ermöglicht außerdem eine übersichtliche Sortierung im Dateibrowser bei einfacher Sortierung nach Namen.

Als Format für den Inhalt der Messwertdateien kommt einfacher ASCII-codierter Text zum Einsatz. So wird bestmögliche Lesbarkeit auf verschiedenen Systemen ermöglicht und die

Daten lassen sich in verschiedene Anwendungen leicht zu weiteren Auswertung importieren. Die einzelnen Messwerte sind durch Leerzeichen getrennt.

5 Vergleich der Trackingverfahren

5.1 Versuchsaufbau

Für die Messreihen wird als optischer Wellenleiter ein aramidverstärkter Plastlichtwellenleiter an die Sende- und Empfangsdioden angeschlossen. Die dazugehörige Sende- und Empfangsplatine ist bereits vorhanden und konnte nach einem Abgleich im Modulsystem verwendet werden. Der Wellenleiter wird, um eine kontrollierte Dehnung zu erhalten, in die sogenannte Gitarre eingespannt. Das ist eine einfache Metallgrundplatte, auf die links und rechts der Wellenleiter eingespannt wird, wobei eine Einspannung mittels Mikrometerschraube fein in Längsrichtung der Faser verstellbar ist. Abbildung 5.1 zeigt den hier beschriebenen Aufbau. Im Vordergrund befindet sich die beschriebene Gitarre mit der eingespannten Faser, rechts das Modulsystem und links der Hostrechner.

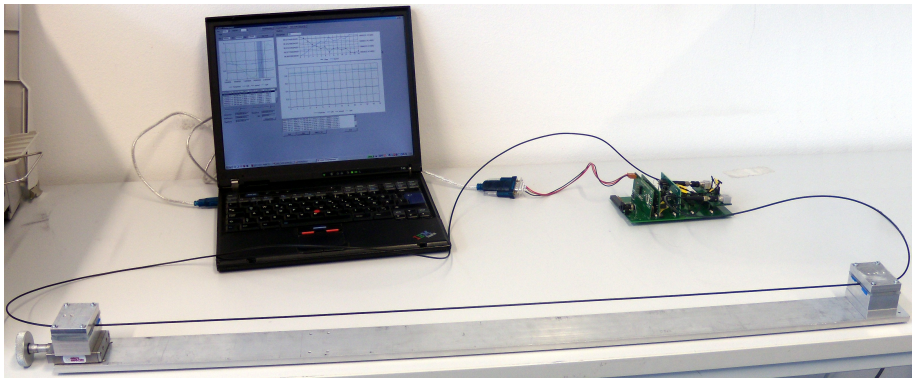

Abbildung 5.1: Komplett aufgebautes Messsystem

Die POF wird nach dem Einspannen noch leicht vorgespannt, so dass man davon ausgehen kann, dass eingestellte Dehnungen an der Mikrometerschraube auch wirklich auf die Faser weiter gegeben werden. Das Modulsystem liegt während der gesammten Messreihen ohne Gehäuse neben den Austrittsöffnungen der Lüftung des Laptops, was eine einigermaßen konstante Temperaturumgebung schaffen sollte und die vorhandenen Temperaturabhängigkeiten der Bauelemente minimiert. Eine weitgehend konstante Spannung des Temperatursensors bei den Messungen bestätigt dies.

Profilmessungen (vgl. Abbildung 5.2) ergeben für die verwendete Faserlänge einen günstigen Arbeitspunkt bei etwa 19 Mhz beim Frequenztracking, da hierbei die Phasenverschiebung bei der Faserlänge bei etwa 90° liegt und das Profil sehr linear ist. Aus Gründen der Vergleichbarkeit wurden alle Phasentrackingmessungen ebenfalls bei einer Frequenz von 19 MHz durchgeführt. Das Profil ist sonst unabhängig von der Frequenz identisch mit

dem Kurvenverlauf von VPHS über die Phasendifferenz des AD8302, wie Abbildung 5.3 zeigt. Die ausführliche Auswertung der Messreihen sowie die Erstellung der Kurvenverläufe erfolgen in OpenOffice Calc.

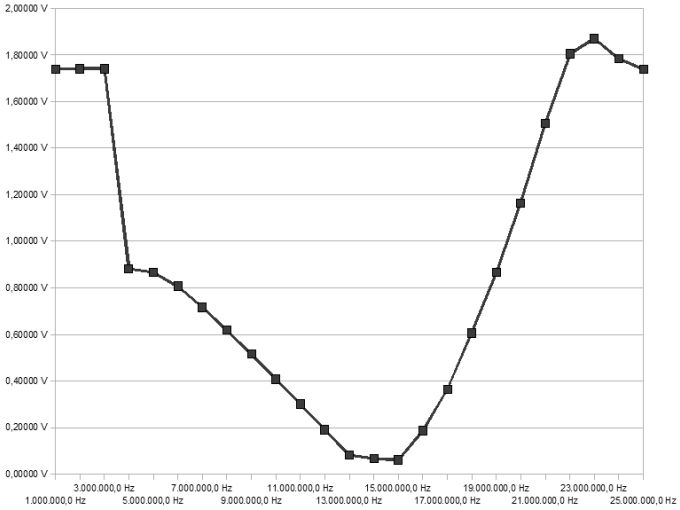

Abbildung 5.2: Ergebnis einer Profilessung, f-Tracking

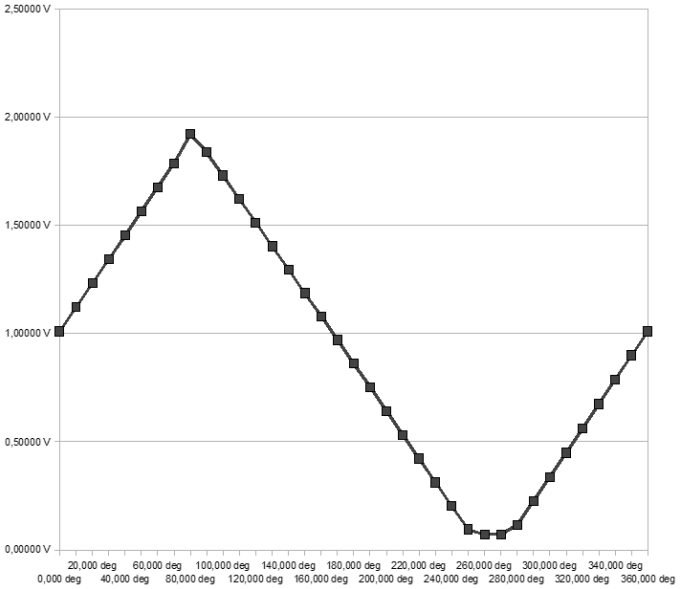

Abbildung 5.3: Ergebnis einer Profilessung, ϕ-Tracking

5.2 Versuchsablauf

Jedes folgend beschriebene Messszenario wird sowohl mit Frequenz- als auch Phasentracking durchgeführt. Im Sinne der Reproduzierbarkeit werden alle Messungen drei mal wiederholt. Dies verringert die Gefahr zufälliger Messkurven und ist zugleich eine Bestätigung für die Ergebnisse.

Im ersten Szenario verbleibt die Faser nun für zwei Minuten in Ruhestellung und wird dann um zwei Millimeter gedehnt. In diesem Zustand verbleibt sie dann für fünf Minuten,

um anschließend wieder um die zwei Millimeter entspannt zu werden. In diesem Zustand werden noch für weitere zwei Minuten die Messwerte aufgenommen.

Das zweite Szenario ist vom Verfahren her identisch mit dem ersten, mit dem Unterschied, dass diesmal jeder Messpunkt zehn mal gemittelt wird. Dies sorgt für einen ruhigeren Verlauf der Messkurve und verringert den Einfluss von Störungen.

Als zusätzliches Szenario wird die Faser alle zehn Sekunden um zwei Millimeter bis zu einer Gesamtdehnung von sechs Millimetern gedehnt. Anschließend wird die Faser wieder in Zwei-Millimeterschritten entlastet. Diese Messungen werden jeweils zehn mal durchgeführt und sind in erster Linie nur eine Bestätigung, dass das Messsystem zuverlässig und reproduzierbar arbeitet.

5.3 Ergebnisse und Auswertung

Die Messfaser zeigt sich in den Versuchen sehr berührungsempfindlich. Dies kommt besonders dann zum Tragen, wenn man beim Verstellen der Mikrometerschraube an die eingespannte Faser gelangt. Sofort verschiebt es die Phase zwischen Ein- und Ausgang, die Regelung läuft nach und man hat einen sehr starken Ausschlag in der Messkurve. Dies zeigt sich zum Beispiel in Grafik A.1 auf Seite 32 in der blauen Kurve. Die genaue Ursache dafür ist nicht ganz klar, da eine plötzliche Erhöhung der Faserdämpfung aufgrund der ebenfalls gemessenen Dämpfungswerte, welche keine solchen Ausschlag zeigen, ausgeschlossen werden kann. Denkbar wäre hier noch als Ursache, dass durch die plötzliche Änderung des Biegeradius ein Teil der Moden der Faser in äußeren Mantelschichten geführt werden und dadurch längere Laufzeiten erleiden, was wiederum zu Störungen an der Empfängerdiode führt und einen unsauberen Signalverlauf zur Folge hat.

Es ist auf alle Fälle anzuraten, U_ϕ zu jedem Messzeitpunkt mehrmals auszulesen und die Werte zu mitteln. Zum einen minimiert man dadurch Messfehler des AD8302, zum anderen können solche massiven Störungen durch Berühren der Faser reduziert werden. Die anghangenen Grafiken verdeutlichen dies. So ist unabhängig vom Trackingverfahren ein ruhigerer, d.h. störungsfreierer, Messverlauf möglich, wenn die Messwerte gemittelt werden. Zudem lassen sich die Dehnungsstufen deutlicher unterscheiden.

In Abbildung A.3 ist zu sehen, dass kurz nach Dehnung der Faser bei der gelben Kurve die Frequenz kontinuierlich nach oben geht. Die Messung wurde daraufhin abgebrochen. Offenbar ist die Messfaser nicht fest genug eingespannt gewesen und schlussendlich durchgerutscht.

Für weitere, genauere Untersuchungen und Vergleiche wurde auf die Messergebnisse mit Mittlung zurück gegriffen. Um die subjektiv aus den Kurvenverläufen ziehbaren Schlüsse über die Genauigkeit der beiden zu vergleichenden Messverfahren auch mit Zahlen zu belegen, wurde für die jeweils drei Messungen die Standardabweichung der Regelgröße ermittelt. Beim Frequenztracking die eingestellte Frequenz, beim Phasentracking die eingestellte Phasenverschiebung. Um vergleichbare Werte zu erlangen, wurde zunächst bei

jeder Messreihe die Differenz zwischen jedem Regelwert und seinem Vorgänger ermittelt und mit einem Faktor skaliert. Dieser Faktor entspricht dem Sprung in Hertz bzw. Grad, wenn die Faser um zwei Millimeter gedehnt wird. Aus Grafik A.3, Seite 34 lässt sich dieser Faktor für das Frequenztracking mit 2000 Hz quantifizieren, in Grafik A.4, Seite 35 für das Phasentracking mit 0,05 Grad. Die Standardabweichung der skalierten Differenzwerte liefert nun vergleichbare, dimensionslose Zahlen, die das Rauschen um die Ruhelage beinhalten.

5.4 Fazit

Für beide Trackingvarianten existieren unter gleichen Bedingungen, d.h. beide arbeiten im gleichen Frequenzbereich, sowohl Vor- als auch Nachteile. Wie die Werte der Standardabweichung in Tabelle 5.1 zeigen, ist das Frequenztracking hochauflösender und rauschärmer. Das bedeutet, dass die Regelgröße beim Frequenztracking in Ruhelage wesentlich weniger schwankt als beim Phasentracking, was ein Detektieren von Änderungen in der Länge einfacher macht. Unter dieser Betrachtungsweise und der Vernachlässigung weiterer Gesichtspunkte ist das Frequenztracking klar im Vorteil. Dieses Ergebnis wird auch dadurch begünstigt, dass sich die Frequenz beim AD9958 mit 32 Bit viel genauer einstellen lässt als die Phase, die nur ein 14 Bit breites Wort zur Verfügung hat.

	f-Tracking	ϕ-Tracking
1. Messung	0,0795	0,2180
2. Messung	0,0799	0,2207
3. Messung	0,0791	0,2484

Tabelle 5.1: Werte der Standardabweichung

Das Phasentracking hat allgemein den Nachteil, dass es einen höheren Schaltungs- und Softwareaufwand erfordert. Die verwendete Doppel-DDS ist nicht ganz unproblematisch in ihrer Handhabung. Das Schaltungslayout ist nicht ohne weiteres erstellbar, es müssen einige Besonderheiten beachtet werden, und auch die Inbetriebnahme und SPI-Kommunikation klappt nicht auf Anhieb. Hinzu kommt, dass die Doppel-DDS ein recht teures Bauelement ist, was dem Sinn des Modulsystems, nämlich ein kostengünstiges und hinreichend genaues Messsystem zu sein, entgegen steht.

Der Hauptvorteil des Phasentrackings bleibt aber bestehen. Da es bei einer festen Frequenz arbeitet, können Baugruppen wie das Ausgangsfilter, die Sende- und Empfangsverstärker sehr schmalbandig ausfallen, was zum einen die Kosten wieder absenkt, zum anderen die Verwendung empfindlicherer Baugruppen ermöglicht. Ebenso entfällt die Frequenzabhängigkeit des gesamten Messsystems, insbesondere der Messfaser, was zu besseren Ergebnissen führen kann.

Ein weiterer, wichtiger Punkt, der im Rahmen dieser Arbeit aber nicht genauer untersucht wurde, ist der des Ernergieverbrauchs. Da das Modulsystem auch autark mit Batteriebetrieb oder mit Versorgung über PoE arbeiten können soll, ist dieser sehr relevant. Die Doppel-DDS hat eine hohe Stromaufnahme, was dazu führt, dass das Modulsystem mit ihr um die 350 mA verbraucht. Deutlich mehr als mit der einfachen DDS.

6 Abschließende Bemerkungen

Das Ziel der Aufgabenstellung, den Vergleich der beiden Trackingverfahren, wurde erreicht. Es konnte im abschließenden Fazit detailliert auf verschiedene Vor- und Nachteile eingegangen werden und Empfehlungen ausgesprochen werden. Das Modul konnte erfolgreich in Betrieb genommen werden und ein fehlerbereinigtes Layout liegt vor.

Die Software des µC arbeitet zuverlässig und stellt alle notwendigen Funktionen bereit, die Doppel-DDS kann korrekt gesteuert werden. Eine weitgehende Nutzung von Unterfunktionen lässt auch die einfache Weiterverwendung von Programmteilen zu. Die Hostsoftware ist nicht perfekt, bei einer bestimmungsgemäßen Nutzung liefert sie aber zuverlässige und reproduzierbare Messungen.

Was bei dieser Arbeit nicht betrachtet wurde, ist die Ermittlung der absoluten Längenänderung. Dies lies sich nicht ohne weiteres mit logischen Ergebnissen realisieren und wurde, da es für den direkten Vergleich unter gleichen Bedingungen nicht zwingend notwendig ist, ausgelassen. Ebenso wurden Drifts in den Messreihen sowie die bereits erwähnte, starke Empfindlichkeit der Faser gegenüber Berührungen nicht weiter untersucht. Dies hätte den Rahmen der Arbeit gesprengt und war für den grundlegenden Vergleich ebenfalls nicht erheblich.

A Anhang

Im Anhang finden sich die Messkurven von Messungen, der Schaltplan als auch das fertige Layout des Doppel-DDS-Moduls, der Quellcode des Mikrocontrollerprogramms sowie der Quellcode der Hostsoftware. Des weiteren liegt der Diplomarbeit eine CD-ROM bei, die in selbsterklärenden Unterverzeichnissen folgende Daten enthält:

- diese Diplomarbeit als pdf-Datei
- die Schalt- und Layoutpläne des Moduls für EAGLE 5.10 und höher
- die vollständigen Projektdateien des Mikrocontrollerprogramms für MS Visual Studio 2008
- die vollständigen Projektdateien der Hostsoftware
- die Simulationsdateien des Ausgangsfilters für AWR Microwave Office 2002
- die im Literaturverzeichnis benannten, im Internet verfügbaren pdf-Dokumente
- die gesammelten Messreihen als einfache Textdateien sowie
- die in OpenOffice-Calc erstellten Tabellen und Diagramme.

Anhang 1: Grafiken

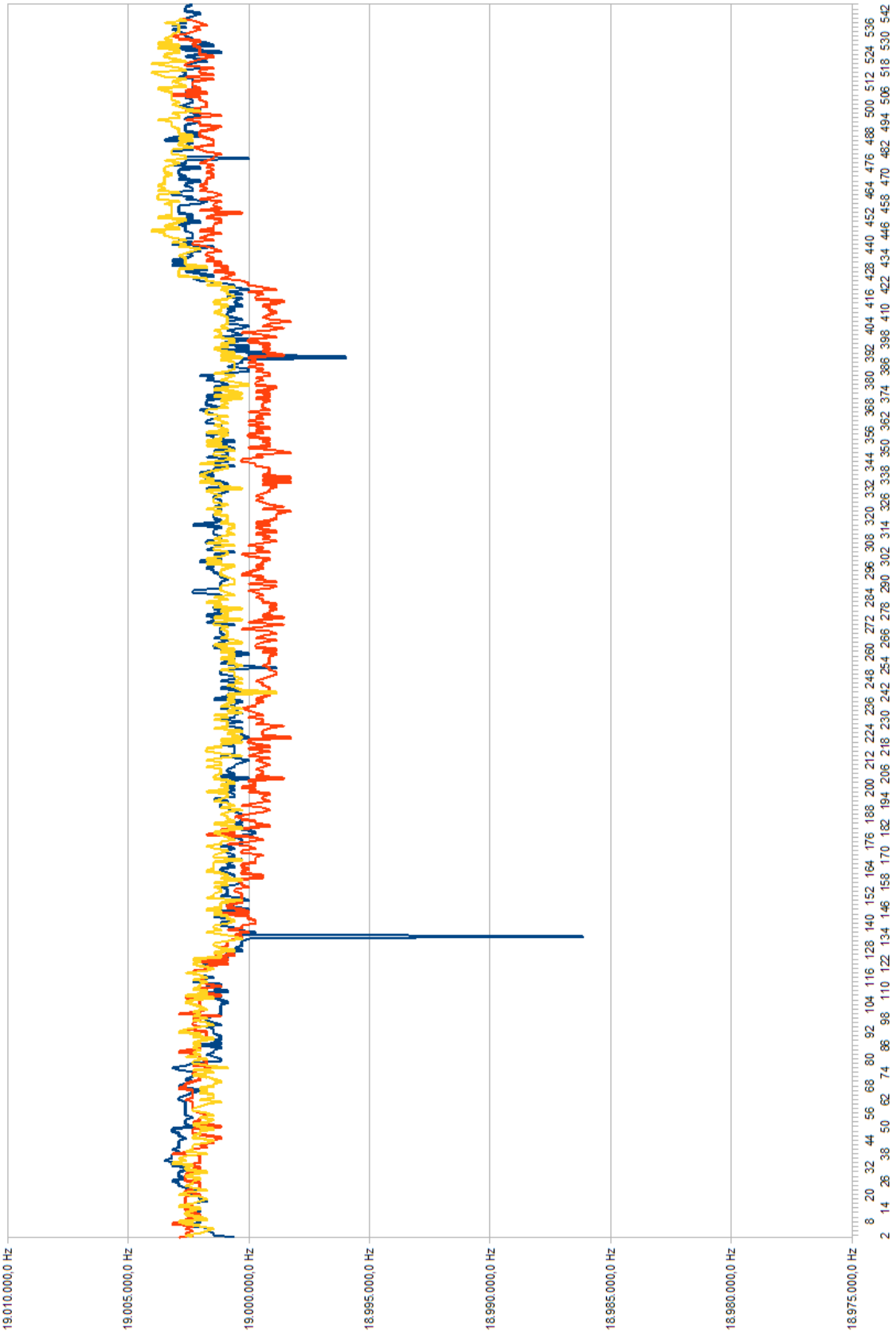

Abbildung A.1: Messergebnisse f-Tracking, ohne Mittlung

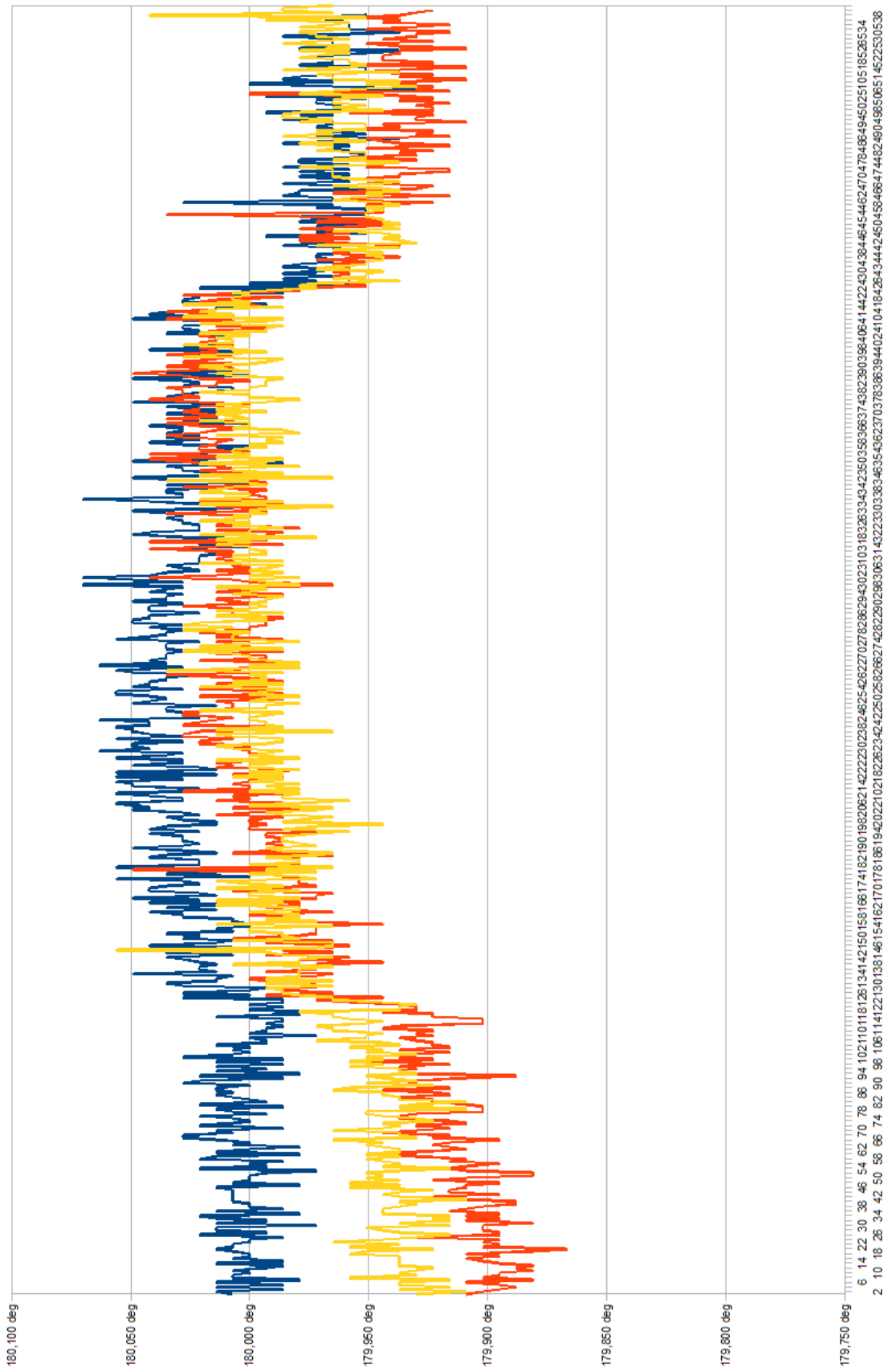

Abbildung A.2: Messergebnisse ϕ-Tracking, ohne Mittlung

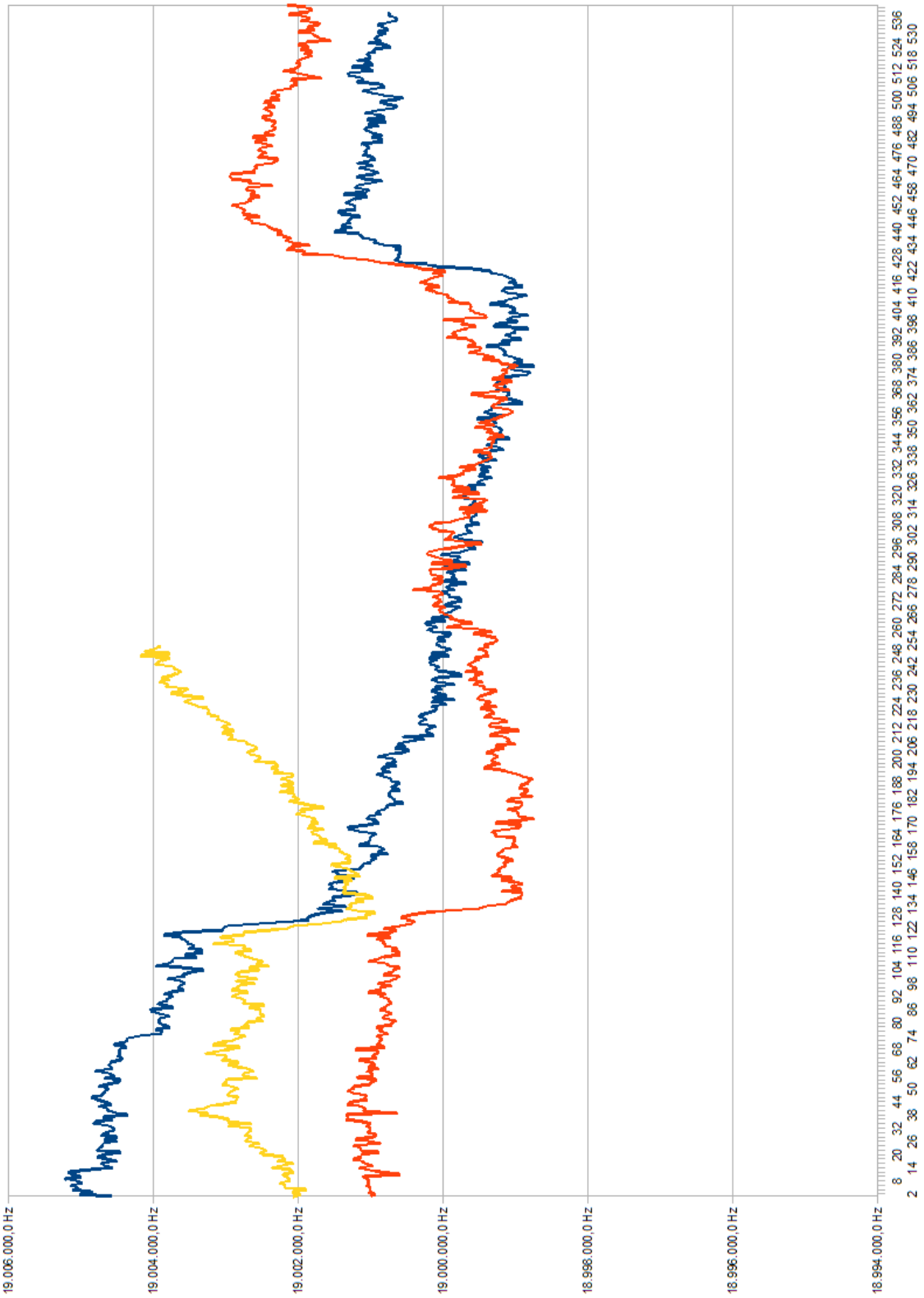

Abbildung A.3: Messergebnisse f-Tracking, mit Mittlung

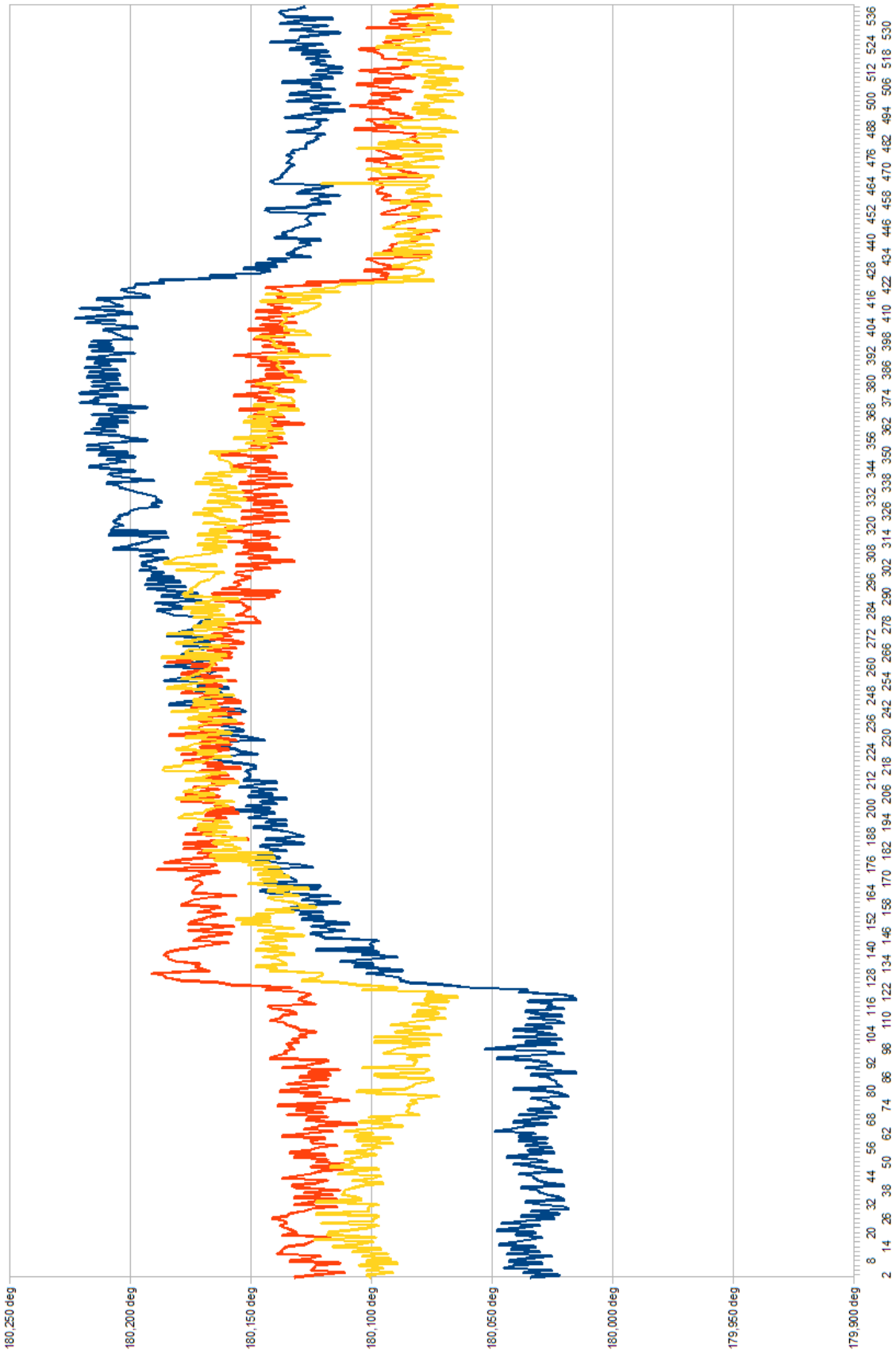

Abbildung A.4: Messergebnisse ϕ-Tracking, mit Mittlung

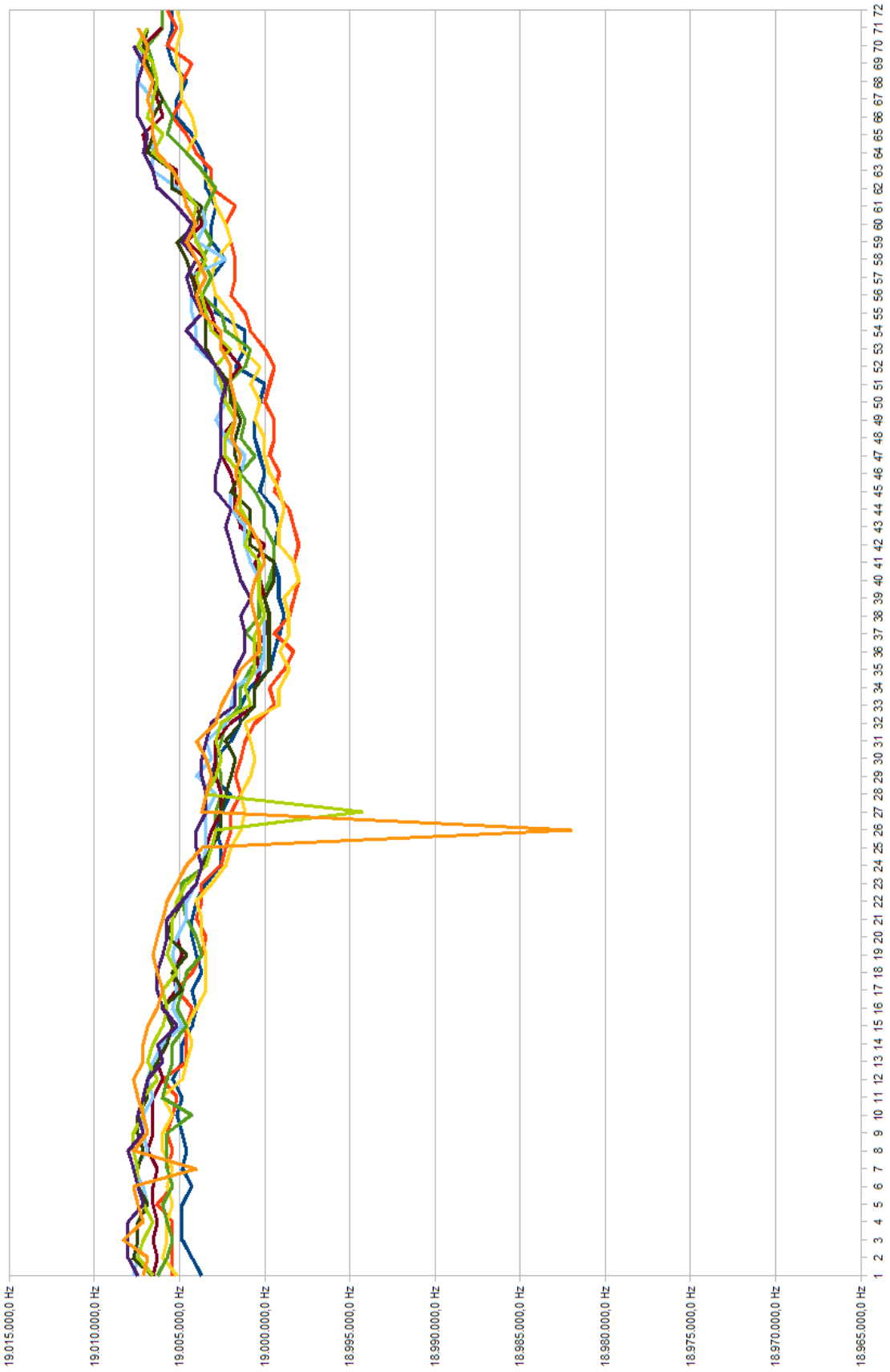

Abbildung A.5: Messergebnisse f-Tracking, ohne Mittlung, 3 Stufen

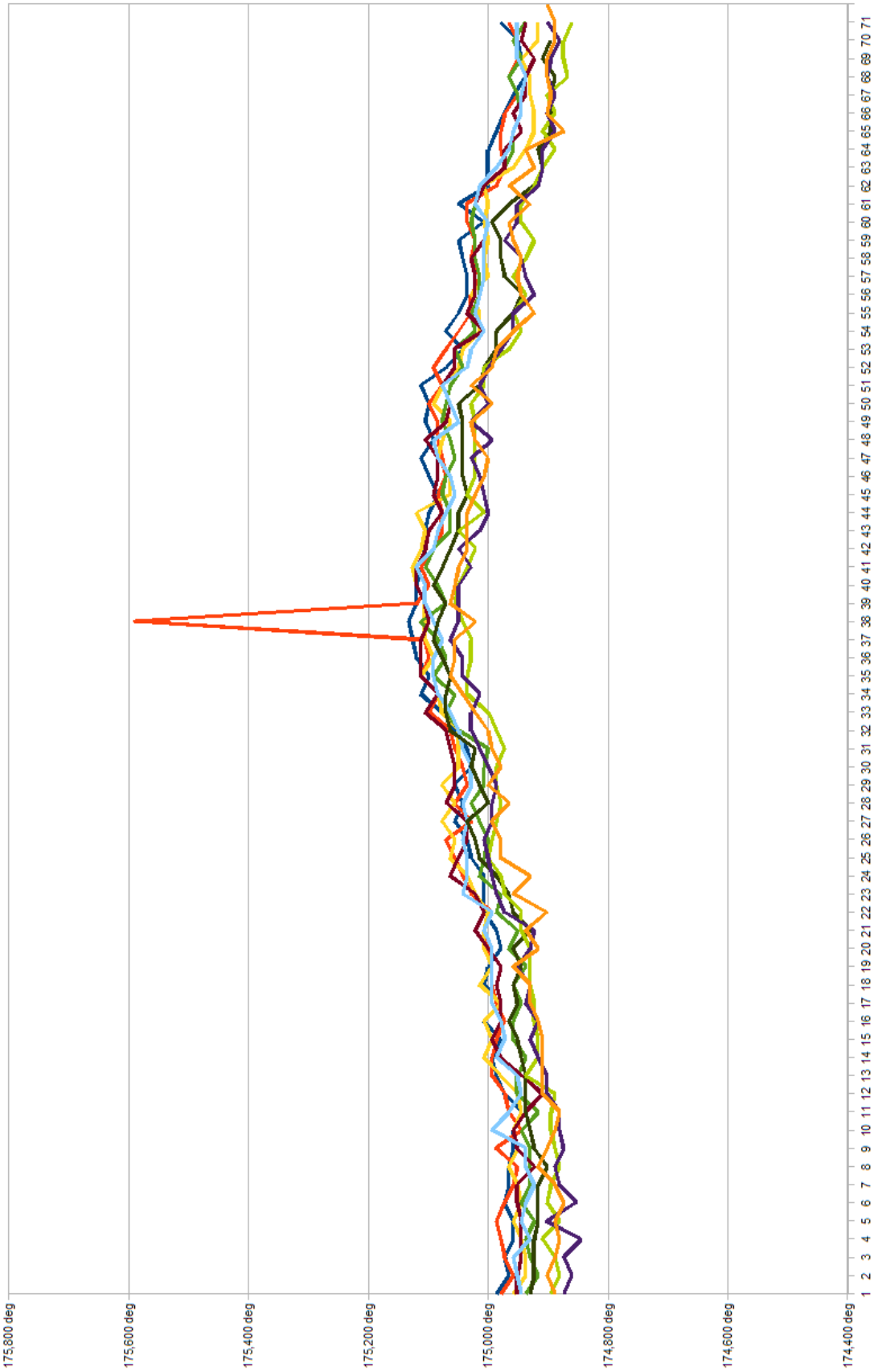

Abbildung A.6: Messergebnisse ϕ-Tracking, ohne Mittlung, 3 Stufen

Anhang 2: Schaltplan Doppel-DDS

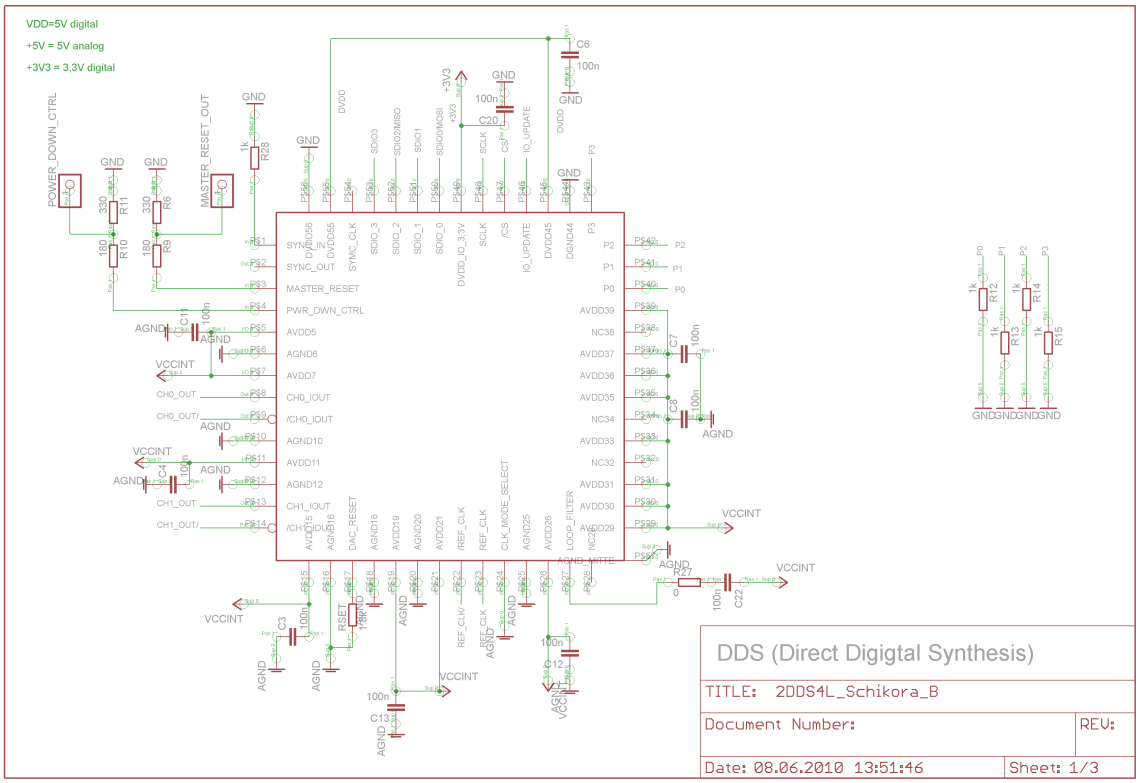

Abbildung A.7: Schaltplan Doppel-DDS-Modul, Seite 1

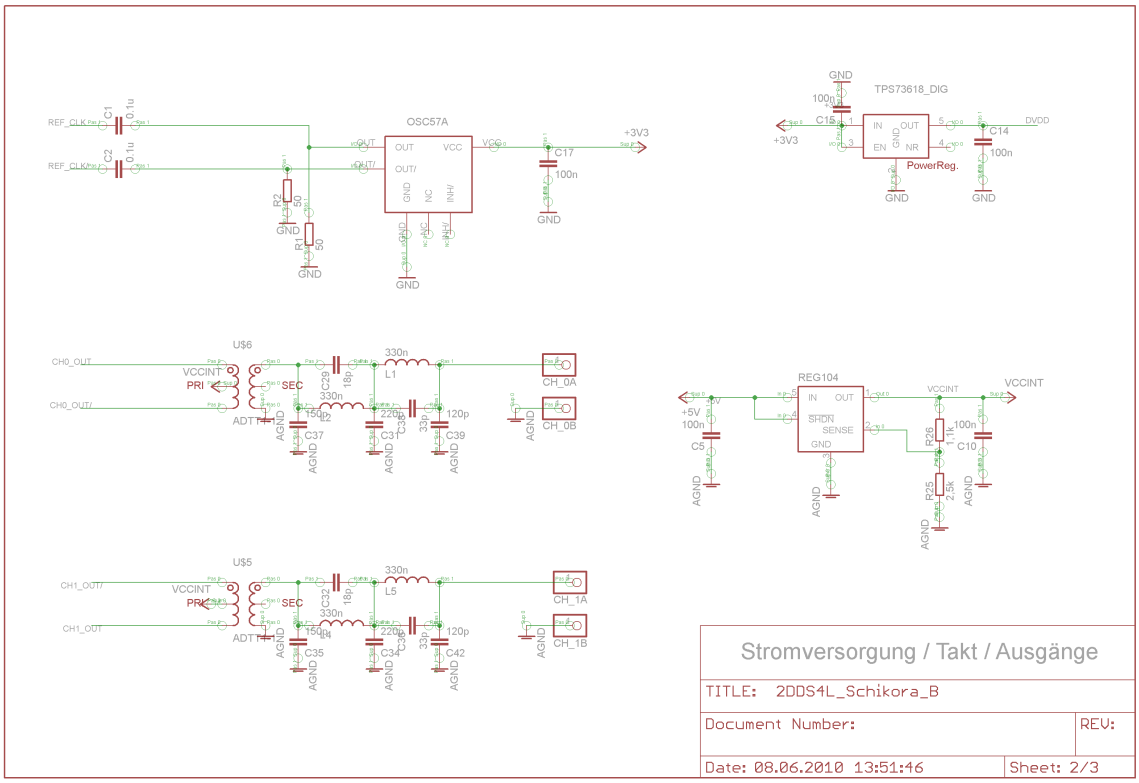

Abbildung A.8: Schaltplan Doppel-DDS-Modul, Seite 2

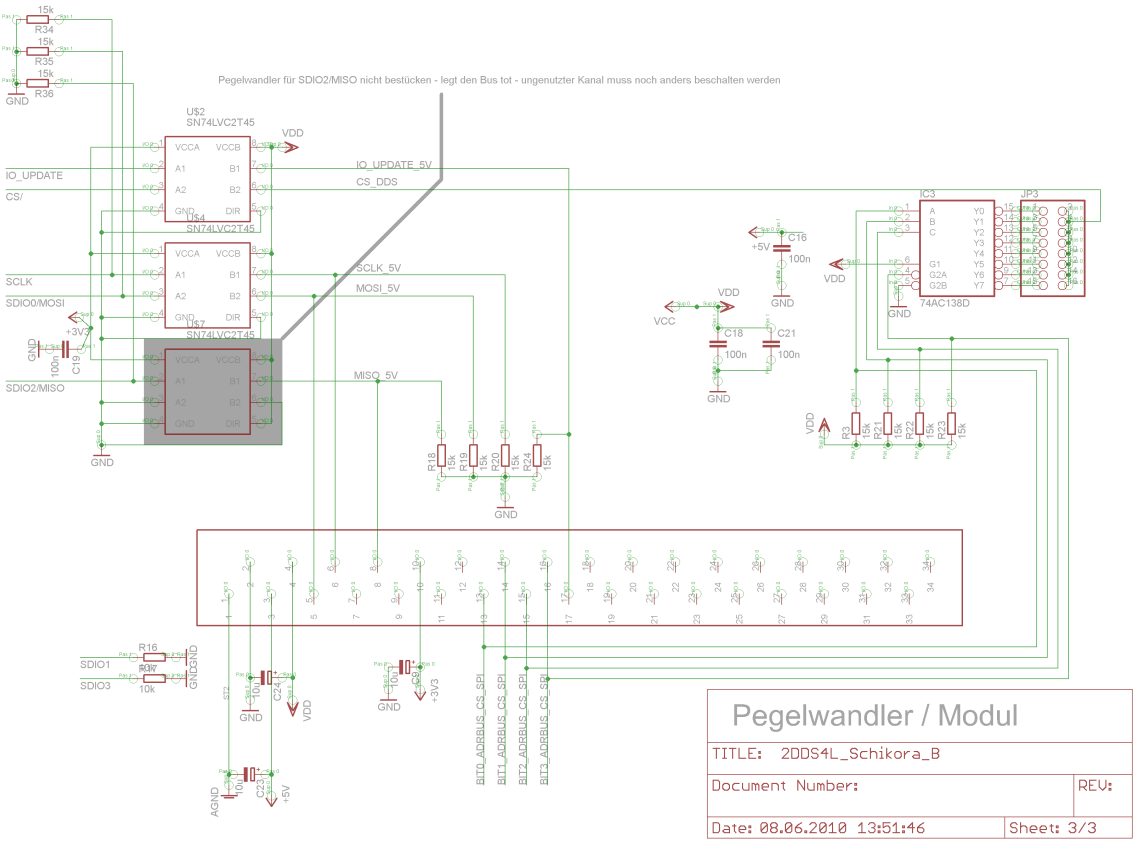

Abbildung A.9: Schaltplan Doppel-DDS-Modul, Seite 3

Anhang 3: Layout Doppel-DDS

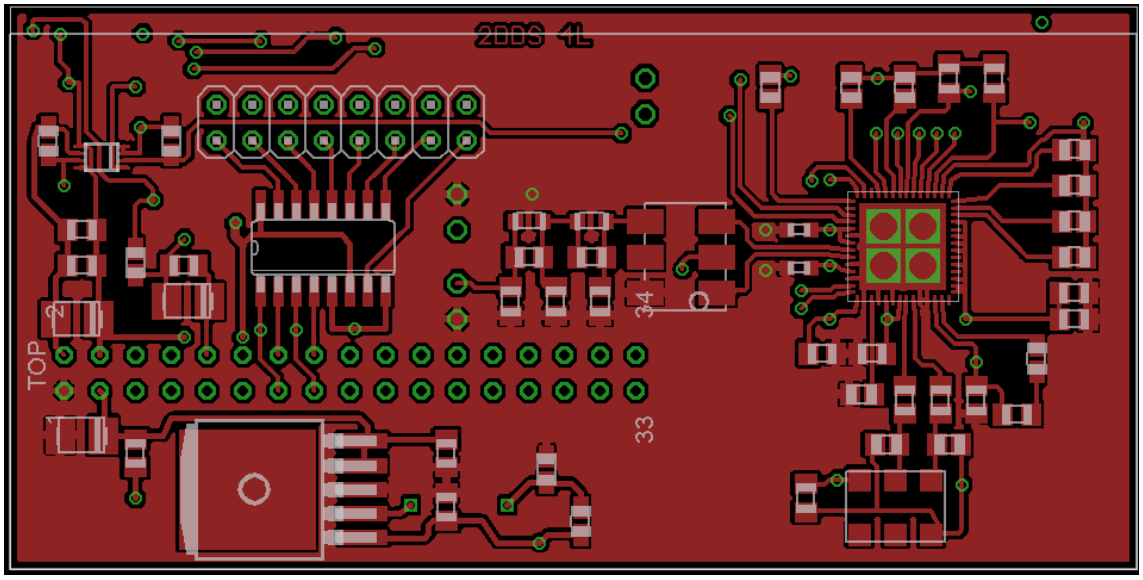

Abbildung A.10: Layout Doppel-DDS-Modul, Vorderseite

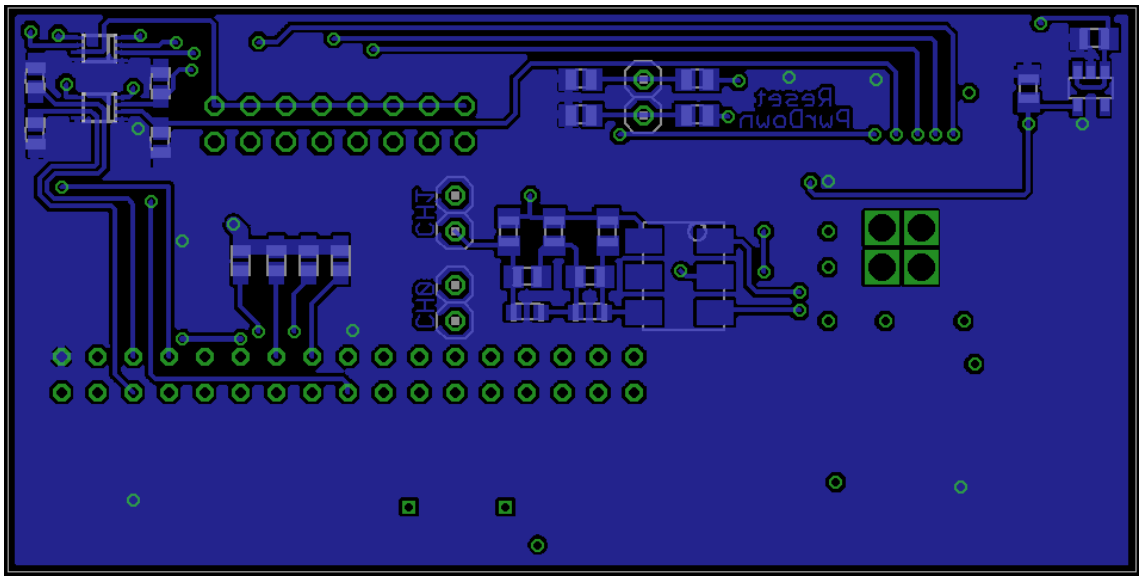

Abbildung A.11: Layout Doppel-DDS-Modul, Rückseite

Anhang 4: Quellcode ATmega128

```
/*************************************************************
Include - Files
*************************************************************/
#include <io.h>
#include <iom128.h> // symbolische Bezeichnung der Register
#include <signal.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <inttypes.h>
#include <spi_cs_adr.h>

/*************************************************************
Definitionen
*************************************************************/
#define CH0 0
#define CH1 1
#define BOTH 2
#define REF_CLK 156250 //in MHz
#define baud 34 //Baudrate 57,6 kBaud (Atmel-Beschr. S.197)
#define MITTLUNGEN 2 //min 2, da erster Messwert verworfen wird


/*************************************************************
Variablen
*************************************************************/
volatile unsigned char testen=0; //nur für Testzwecke
volatile unsigned int ergebnis; //wird in AD_AUSLESEN verwendet
volatile unsigned int result; //enthält das zurückgegebene Ergebnis in ↩
    AD_AUSLESEN
volatile uint32_t PhaseTuningWord;
volatile uint32_t FrequencyTuningWord;
volatile unsigned char SelectedChannel;
unsigned char stw_ascii[12]; //empfangene Steuerworte vom VB Programm
volatile unsigned char usart0_zaehl=0; //Zählvariable für einkommende Befehle ↩
    über USART Schnittstelle
volatile int test=0;
unsigned char a=0; //Zählvariable um empf. Frequenzwort zusammenzusetzen
int steuerwort; //C-Fkt. atoi verlangt 16 Bit Variable (verwendet in ↩
    STRING_ZU_INT() )
volatile unsigned char kanal; //Kanalauswahl bei ADC
unsigned int wandlungs_erg; //Variable enthält Ergebnis des ADC (AD_AUSLESEN() )


/*************************************************************
STATUSFLAGS
*************************************************************/
volatile struct //selbst definiertes Statusregister
{
    unsigned char AD_BEFEHL: 1; //wird gesetzt wenn 2 Byte über USART0 empfangen ↩
        wurden (decodiert 00 bis 09)
    unsigned char DDS_BEFEHL_6BYTE_EMPF :1;
    unsigned char UEB_FEHLER: 1;
} STATUSFLAGS;

/*************************************************************
Funktionsprototypen
```

```
54 ********************************************************/
55 void ALLG_INIT(void); //allgemeine Initialisierungen
56 void SPI_INIT(void); //initialisiert SPI-Interface
57 void SPI_CS_INIT(void); //Initialisierung der CS-Leitungen für SPI Komponenten
58 void warten(unsigned int);
59 void DDS_INIT(void); //Initialisierung der Doppel-DDS AD9958
60 void SPI_2DDSSend(unsigned char, unsigned char, uint32_t); //SPI-Uebertragung für↩
       Doppel-DDS
61 void SPI_SEND(unsigned char); //sendet 1 Byte auf den SPI Bus
62 void IO_Update(void); //I/O-Update an die DDS
63 unsigned int AD_AUSLESEN(unsigned char, unsigned char); //AD-Wandler auslesen
64 void SET_FREQUENCY(unsigned char, uint32_t); //Frequenz der 2DDS setzen, in MHz
65 void SET_PHASE(unsigned char, uint32_t); //Phase der 2DDS setzen
66 void CHANNEL_SELECT(unsigned char); //Kanalauswahl 2DDS
67 void INT_INIT(void); //Interrupt initialisieren
68 void USART_INIT(unsigned int); //serielles Interface initialisieren
69 void USART_SEND(int);
70 void USART0_SENDESTRING(unsigned char*);
71 void USART0_SENDBYTE(unsigned char);
72 int STRING_ZU_INT(unsigned char*);
73
74
75 /*********************************************************
76 Hauptprogramm
77 *********************************************************/
78 int main(void) //angepasst
79 {
80     SREG = 0x00;
81
82     DDRA |= (1<<DDA1);
83     PORTA |= (1<<PA1);
84     warten(1000);
85     PORTA &= ~(1<<PA1);
86     warten(1000);
87
88     DDRC |= (1<<DDC7);
89     PORTC &= ~(1<<PC7);
90     warten(1000);
91     PORTC |= (1<<PC7);
92     warten(1000);
93
94     ALLG_INIT();
95     SPI_INIT();
96     SPI_CS_INIT();
97     DDS_INIT();
98     USART_INIT(baud);
99     INT_INIT();
100
101    while(1)
102    {
103        if (STATUSFLAGS.UEB_FEHLER == 1) //Übertragungsfehler auf USART0 ↩
               aufgetreten (erstes
104        {                    //Hex-Zeichen ist weder 30 noch 31 (0 oder 1))
105            STATUSFLAGS.UEB_FEHLER = 0; //Zeichen sind ASCII codiert (für 4 Bit ↩
                   Hex-Zeichen
106                          //werden 8 Bit übertragen
107            USART0_SENDBYTE('?'); //Fehler senden
108
109            for (a=0; a<13; a++) //stw_ascii löschen
110            {
111                stw_ascii[a]=0;
112            }
```

```
113         }
114         //-------------------------------2DDS Befehl-----------------------
115         if (STATUSFLAGS.DDS_BEFEHL_6BYTE_EMPF == 1)
116         {
117             SREG &= 0x7f;
118             if ((stw_ascii[1] & 0x04) && (stw_ascii[1] & 0x08)) //Steuerbyte ↩
                    prüfen
119             {
120                 STATUSFLAGS.UEB_FEHLER = 1;
121             }
122             else
123             {
124                 if ((stw_ascii[1] & 0x01) && (stw_ascii[1] & 0x02)) //↩
                        Kanalauswahl auslesen
125                 {
126                     SelectedChannel = BOTH; //beide Kanäle
127                 }
128                 else
129                 {
130                     if (stw_ascii[1] & 0x01)
131                     {
132                         SelectedChannel = CH0;  //Kanal 0
133                     }
134                     else
135                     {
136                         if (stw_ascii[1] & 0x02)
137                         {
138                             SelectedChannel = CH1; //Kanal 1
139                         }
140                     }
141                 }
142                 if (stw_ascii[1] & 0x04) //Frequenzwort übertragen
143                 {
144                     FrequencyTuningWord = 0;
145                     for (a=2; a<=5; a++) //zusammen setzen
146                     {
147                         FrequencyTuningWord = FrequencyTuningWord | stw_ascii[a];
148                         if (a < 5) FrequencyTuningWord = FrequencyTuningWord << ↩
                                8;
149                     }
150                     SET_FREQUENCY(SelectedChannel, FrequencyTuningWord); //↩
                            Unterfunktion aufrufen
151                 }
152                 else
153                 {
154                     if (stw_ascii[1] & 0x08) //Phasenwort
155                     {
156                         PhaseTuningWord = stw_ascii[4];
157                         PhaseTuningWord = PhaseTuningWord << 8;
158                         PhaseTuningWord = PhaseTuningWord | stw_ascii[5];
159                         SET_PHASE(SelectedChannel, PhaseTuningWord);
160                     }
161                 }
162             }
163             STATUSFLAGS.DDS_BEFEHL_6BYTE_EMPF = 0;
164             for (a=0; a<13; a++)    //stw_ascii löschen
165             {
166                 stw_ascii[a]=0;
167             }
168             SREG |=0x80;
169         }
170         //-------------------------------AD-Befehl-------------------------
```

```
171         if (STATUSFLAGS.AD_BEFEHL == 1) //STATUSFLAGS.AD_BEFEHL ist eins wenn 2 ↩
                Byte eingelesen wurden
172         {
173             STATUSFLAGS.AD_BEFEHL = 0; //AD-Befehls-Erkennungs-Bit zurücksetzen
174             steuerwort=STRING_ZU_INT(stw_ascii); //Fkt. verlangt 16Bit Variable (↩
                    steuerwort)
175 
176             if (steuerwort == 1 || steuerwort ==2 || steuerwort ==3 || steuerwort↩
                     ==4)
177             {
178                 switch (steuerwort)
179                 {
180                     case 1:
181                         kanal=1;
182                         break;
183                     case 2:
184                         kanal=2;
185                         break;
186                     case 3:
187                         kanal=3;
188                         break;
189                     case 4:
190                         kanal=4;
191                         break;
192                     default:
193                         break;
194                 }
195                 wandlungs_erg = AD_AUSLESEN(kanal,MITTLUNGEN); // Kanal 4 -> ↩
                        Phase; Kanal 2 -> Amplitude
196                 USART_SEND(wandlungs_erg);
197                 USART0_SENDBYTE(0x0d); //CR (Wagenrücklauf)
198                 USART0_SENDBYTE(0x0a); //LF (Line Feed)
199             }
200             for (a=0; a<13; a++)    //stw_ascii löschen
201             {
202                 stw_ascii[a]=0;
203             }
204         }
205         //---------------------------------------------------------------
206     } //while(1)
207 } //main
208 
209 /*****************************************************
210 Funktionen
211 *****************************************************/
212 //===============================================================
213 void ALLG_INIT(void) //angepasst
214 {
215     DDRD |= (1<<DDD4); //PD4 als Ausgang festlegen -> LED - HMI
216     DDRD |= (1<<DDD5); //PD5 als Ausgang festlegen -> LED - HMI
217     DDRD |= (1<<DDD6); //PD6 als Ausgang festlegen -> LED - HMI
218     DDRD |= (1<<DDD7); //PD7 als Ausgang festlegen -> LED - HMI
219     DDRA |= (1<<DDA5); //PA5 als Ausgang festlegen (A0 Leitung ADC)
220     DDRA |= (1<<DDA4); //PA4 als Ausgang festlegen (A0 Leitung ADC)
221     DDRA  |= (1<<DDA2); //Pin 2 an Port A auf Ausgang programmieren (R/C Signal)
222     DDRA  &=~(1<<DDA3); //Pin 3 an Port A auf Eingang programmieren (BUSY Signal)
223     DDRA |= (1<<DDA7); //zur aktivierung des I/O-Update an der 2DDS
224     PORTA &= ~(1<<PA7); //I/O-Update nullen
225 }
226 //===============================================================
227 void SPI_INIT(void) //angepasst
228 {
```

```
229     DDRB |= (1<<DDB0); //PB0 (SS NICHT) als Ausgang programmieren
230     PORTB |= (1<<PB0);
231     DDRB |= (1<<DDB1); //SCK muss als Ausgang definiert werden um Clock abzugeben
232     DDRB |= (1<<DDB2); //MOSI als Ausgang definiert
233     DDRB &= ~(1<<DDB3);
234     SPCR &= ~(1<<DORD); //MSB vom Datenwort wird zuerst gesendet
235     SPCR |= (1<<MSTR); //uC im Master Mode
236     SPCR |= (1<<CPOL); //SCK ist 1 wenn nichts zu tun ist
237     SPCR |= (1<<CPHA); //bei steigender Flanke Daten übernehmen
238     SPCR |= (1<<SPR1); //SCK=fOsc/128
239     SPCR |= (1<<SPR0); //SCK=fOsc/128
240     SPSR &= ~(1<<SPI2X); //SCK=fOsc/128
241     SPCR |= (1<<SPE); //SPI Enable
242 }
243 //=============================================================
244 void warten(unsigned int anzahl) //übernomen
245 {
246     unsigned int b,c=0;
247     for (b=0; b < anzahl; b++)
248     c++;
249 }
250 //=============================================================
251 void DDS_INIT(void) //neu - Init der 2DDS
252 {
253     SPI_ADR(5);
254     SPI_2DDSSend(0x00, 1, 0b11000010); //CSR setzen
255     SPI_2DDSSend(0x01, 3, 0b000000000000000000000000); //FR1 setzen
256     SPI_2DDSSend(0x02, 2, 0b0000000000000000); //FR2 setzen
257     SPI_2DDSSend(0x00, 1, 0b11000010);
258     SPI_2DDSSend(0x03, 3, 0b000000000000001100000000); //CFR setzen (DAC Full ↩
            Scale)
259     SPI_2DDSSend(0x00, 1, 0b11000010);
260     SPI_2DDSSend(0x04, 4, 0x28F5C28F); //CTW0 setzen (Frequenzwort, 25MHz)
261     PhaseTuningWord = 0x0000;
262     SPI_2DDSSend(0x00, 1, 0b0100010);
263     SPI_2DDSSend(0x05, 2, PhaseTuningWord); //CPW0 setzen (Phasenwort)
264     SPI_2DDSSend(0x00, 1, 0b10000010);
265     SPI_2DDSSend(0x05, 2, 0x0000); //CPW0 setzen (Phasenwort)
266     SPI_2DDSSend(0x00, 1, 0b11000010);
267     SPI_2DDSSend(0x06, 2, 0x00); //ACR setzen
268     SPI_2DDSSend(0x00, 1, 0b11000010);
269     SPI_2DDSSend(0x07, 2, 0); //LSR setzen
270     SPI_2DDSSend(0x00, 1, 0b11000010);
271     SPI_2DDSSend(0x08, 4, 0); //RDW setzen
272     SPI_2DDSSend(0x00, 1, 0b11000010);
273     SPI_2DDSSend(0x09, 4, 0); //FDW setzen
274     IO_Update(); //issue I/O-Update
275     SPI_ADR(0);
276 }
277 //=============================================================
278 void SPI_2DDSSend(unsigned char addr, unsigned char NumoBytes, uint32_t value)
279 { //neu - überträgt mehrere Bytes an die 2DDS
280     unsigned char sendvalue;
281     SPDR=addr;
282     while(!(SPSR & (1<<SPIF)));
283     do
284     {
285         NumoBytes--;
286         sendvalue=value>>(NumoBytes*8);
287         SPDR=sendvalue;
288         while(!(SPSR & (1<<SPIF)));
289     }
```

```
290     while(NumoBytes!=0);
291 }
292 //=============================================================
293 void IO_Update(void) //neu - IO_Update für die 2DDS
294 {
295     PORTA |= (1<<PA7);
296     warten(100);
297     PORTA &= ~(1<<PA7);
298 }
299 //=============================================================
300 unsigned int AD_AUSLESEN(unsigned char kanal, unsigned char mittlungen)
301 { //übernommen
302     volatile unsigned char in, a;
303     volatile unsigned long average=0; //Variable um zu Mitteln =(ergebnis+↩
        ergebnis+...)/MITTLUNGEN
304
305     for (a=0; a < mittlungen; a++)
306     {
307         ergebnis=0x00;
308
309         if (kanal == 1) //!!!!Achtung: Kanalauswahl im neuen Modulsystem PA4, PA5
310             {
311                 PORTA &= ~(1<<PA4); PORTA &= ~(1<<PA5); //A1,A0 = 0,0 -> Kanal 1 ↩
                    auswählen (Temperatursensor)
312             }
313         if (kanal == 2)
314             {
315                 PORTA |= (1<<PA4); PORTA &= ~(1<<PA5); //A1,A0 = 0,1 -> Kanal 2 ↩
                    auswählen (Amplitude)
316             }
317         if (kanal == 3)
318             {
319                 PORTA &= ~(1<<PA4); PORTA |= (1<<PA5); //A1,A0 = 1,0 -> Kanal 3 ↩
                    auswählen
320             }
321         if (kanal == 4)
322             {
323                 PORTA |= (1<<PA4); PORTA |= (1<<PA5); //A1,A0 = 1,1 -> Kanal 4 ↩
                    auswählen (Phase)
324             }
325
326         SPI_ADR(1); //ADC auswählen -> hat ADR 1
327         PORTA &=~(1<<PA2); // R/C auf 0 legen  -> Wandlung starten
328
329         //********warten bis Wandlung fertig ist (bis BUSY auf HIGH zurückkehrt)↩
                ********
330         do
331         {
332             in = PINA; //ADC benötigt rund 4us Wandlungszeit
333             in &= 0x08; //nur Pin4 an Port A zur Weiterverarbeitung heranziehen
334         }
335         while (in != 0x08);
336         //************************************************************
337
338         PORTA |= (1<<PA2); // R/C auf 1 legen  -> Wandler auslesen
339         SPI_SEND(0x00); //es wird scheinbar über die MOSI Schnittstelle etwas
340                         //gesendet -> siehe SPI Schnittstelle Aufbau (Ringschieber)
341                         // danach Abarbeitung ISR -> höherwertigen Bits lesen
342
343         ergebnis = SPDR;
344         ergebnis = ergebnis<<8; //16 Bit Ergebnis zusammensetzen
345
```

```
346         SPI_SEND(0x00); // niederwertigeren 8 Bit lesen
347         ergebnis |= SPDR; // 16 Bit Ergebnis zusammensetzen
348 
349         SPI_ADR(0); //CS zurücksetzen
350         if (a != 0)  //ersten Messwert verwerfen, da Wandlungsfehler auftreten
351             average=average+ergebnis;
352     }
353     result=average/(mittlungen-1);
354     return result;
355 }
356 //=============================================================
357 void SPI_SEND(unsigned char daten)
358 { //übernommen
359     SPDR = daten;
360     while(!(SPSR & (1<<SPIF))); //wait for transmission complete
361 }
362 //=============================================================
363 void CHANNEL_SELECT(unsigned char channel)
364 { //neu - Kanalauswahl für 2DDS
365     switch (channel)
366     {
367         case CH0:  SPI_2DDSSend(0x00, 1, 0b01000010);
368                    break;
369         case CH1:  SPI_2DDSSend(0x00, 1, 0b10000010);
370                    break;
371         case BOTH: SPI_2DDSSend(0x00, 1, 0b11000010);
372                    break;
373     }
374 }
375 //=============================================================
376 void SET_FREQUENCY(unsigned char channel, uint32_t FTW)
377 { //neu - Frequenz 2DDS setzen
378     SPI_ADR(5);
379     CHANNEL_SELECT(channel);
380     SPI_2DDSSend(0x04, 4, FTW);
381     IO_Update();
382     SPI_ADR(0);
383 }
384 //=============================================================
385 void SET_PHASE(unsigned char channel, uint32_t PTW)
386 { //neu - Phase 2DDS setzen
387     SPI_ADR(5);
388     CHANNEL_SELECT(channel);
389     SPI_2DDSSend(0x05, 2, PTW);
390     IO_Update();
391     SPI_ADR(0);
392 }
393 //=============================================================
394 void USART_INIT(unsigned int baudrate)
395 { //übernommen
396     UBRR0H = (unsigned char)(baudrate>>8); //Baudrate einstellen
397     UBRR0L = (unsigned char)baudrate;
398     UCSR0B = (1<<RXEN0)|(1<<TXEN0); //Freigabe Receiver, Transmitter
399     UCSR0A |= (1<<U2X0); //Double Speed Mode
400     UCSR0C = (1<<UCSZ01)|(1<<UCSZ00); //Einstellung Frame Format: 8 Daten, 1 Stop↩
          Bit
401     UCSR0B |= (1<<RXCIE0); //Freigabe Receive Interrupt
402 }
403 //=============================================================
404 void INT_INIT(void)
405 { //übernommen
406     SREG  |= 0x80; // globale Interrupt-Freigabe -> siehe Programmiermodell S.7
```

```
407 }
408 //=============================================================
409 void USART_SEND(int zahl)
410 { //übernommen
411     char a,b,c=4,versch,anz=0;
412     char *cptr1,*cptr2;
413     char CharBuffer1[8],CharBuffer2[8]; //Puffer so lang, wie Anzahl nötiger
414                         //Stellen + 1 für Endezeichen
415     cptr2=CharBuffer2;
416     for (a=0; a<6; a++) //CharBuffer2 mit 0 (ASCII 30) vorinitialisieren
417     {
418         *cptr2=0x30;
419         cptr2++;
420     }
421
422     itoa(zahl,CharBuffer1,16); //16->als Hexzahl darstellen
423             // !! zahl muss Integer Zahl sein, d.h. Variable ist ↩
                    Vorzeichenbehaftet (-32000...+32000)
424             // bei mir sind Wandlungsergebnisse vorzeichenlos -> unsigned int ↩
                    wandlungs_erg; (0..FFFF)
425             // beim Aufruf der Funktion entsteht Konvertierung von unsigned int ↩
                    nach in,
426             // eigentlich nicht ganz korrekt, aber funktioniert
427             //--------Ermittlung Anzahl der Stellen nach Wandlung von int nach ↩
                    ACSII-----
428             //--------(es müssen 4 Stellen sein, um immer einheitliches ↩
                    Datenpaket an den PC zu senden)
429
430     cptr1=CharBuffer1; //Pointer cprt1 zeigt auf erstes Feldelement von ↩
        CharBuffer1
431     while (*cptr1 != 0x00)
432     {
433         anz++; //Ermittlung Anzahl der Stellen nach Wandlung zu ASCII
434         cptr1++;
435     }
436
437     versch = c-anz;
438     cptr1=CharBuffer1;
439     cptr2=CharBuffer2;
440
441     for (b=0; b<=anz; b++)
442     {
443         *(cptr2+versch)=*cptr1;
444         cptr1++;
445         cptr2++;
446         }
447
448     USART0_SENDESTRING(CharBuffer2);
449 }
450 //=============================================================
451 void USART0_SENDESTRING(unsigned char* data)
452 { //übernommen
453     while (*data)
454     {
455         while (!(UCSR0A & (1<<UDRE0))); //warte bis Transmit Puffer leer ist
456         UDR0 = *data++;  //Daten in den Puffer schreiben (startet die Übertragung↩
                )
457     }
458 }
459 //=============================================================
460 void USART0_SENDBYTE(unsigned char data)
461 { //übernommen
```

```
462     while ( !(UCSR0A & (1<<UDRE0))); //wait for empty transmit buffer
463     UDR0=data; //Put data into buffer, sends the data
464 }
465 //=============================================================
466 int STRING_ZU_INT(unsigned char* data)
467 { //übernommen
468     volatile int ergebnis=0;
469     ergebnis=atoi(data);
470     return ergebnis;
471 }
472 /*********************************************************
473 Interrupt Service Routinen
474 *********************************************************/
475
476 //INTERRUPT(SIG_...) -> aktuelle Abarbeitung der ISR wird durch neuen Interrupt ↩
        unterbrochen
477 //SIGNAL(SIG_...) -> aktuelle Abarbeitung der ISR wird durch neuen Interrupt ↩
        NICHT unterbrochen
478
479 //=============================================================
480 SIGNAL(SIG_USART0_RECV) //wird ausgelöst wenn ein Zeichen (8Bit) empfangen wurde
481 {               //RXC Flag wird gesetzt
482                 //nur Zeichen zwischen 30 und 39 werden richtig gewandelt
483     //angepasst an neue 2DDS-befehle
484     stw_ascii[usart0_zaehl]=UDR0; // !!höherwertigen Bits zuerst empfangen!!
485     usart0_zaehl++;
486
487     if (stw_ascii[0] < 0x30 || stw_ascii[0] > 0x35)
488     {
489         usart0_zaehl=0;
490         STATUSFLAGS.UEB_FEHLER=1;
491         test++;
492     }
493     else if (usart0_zaehl == 2 && stw_ascii[0] == 0x30) //0x30 ist ASCII-↩
            decodiert 0
494     {                   //Steuerworte mit 0 beginnend sind AD-Befehle
495         usart0_zaehl=0; //wenn 2 Byte eingelesen wurden, dann Variable ↩
                zurücksetzen
496         STATUSFLAGS.AD_BEFEHL=1; //und Verarbeitung im Hauptprogramm starten
497         STATUSFLAGS.UEB_FEHLER=0;
498         STATUSFLAGS.DDS_BEFEHL_6BYTE_EMPF=0;
499         test++;
500     }
501     else if (usart0_zaehl == 6 && stw_ascii[0] == 0x35) //0x31 ist ASCII-↩
            decodiert 5
502     {                   //Steuerworte mit 5 beginnend sind 2DDS-Befehle
503         usart0_zaehl=0; //wenn 2 Byte eingelesen wurden, dann Variable ↩
                zurücksetzen
504         STATUSFLAGS.DDS_BEFEHL_6BYTE_EMPF=1; //und Verarbeitung im Hauptprogramm ↩
                starten
505     }
506     else if (usart0_zaehl > 5)
507     {
508         usart0_zaehl=0;
509         STATUSFLAGS.UEB_FEHLER=1;
510         test++;
511     }
512 }
```

Anhang 5: Quellcode Hostsoftware

```
'#############################################################################
'#                                                                           #
'#             Längensensor-Software für die DDS AD9958                      #
'#                                                                           #
'#############################################################################
'
'    Version vom 18.08.2010
'    Autor: Christian Schikora
'
'    nach Vorlage von Prof. Döring
'    (ursprüngliche Profilmessung und DDS-Steuerworterstellung, COM-Kommunikation)
'
'ACHTUNG:
'einfache Testsoftware zum Zwecke der vergleichenden Messungen, umschhaltbar
'zwischen Phasen- und Frequenztracking. Keine besonderen Fehlerbehandlungen,
'keine Überprüfung auf Logik der Benutzereingaben. Überlegtes Vorgehen erforder-
'lich um sinnvolle Messungen zu erhalten. (z.b. nach Umschalten des Trackingver-
'fahrens neue Profilmessung durchführen etc.)

Imports System.IO

Public Class Form1

'###Strukturen################################################################
Private Structure KurvenStruktur
    Dim WY() As Double                                  '* Interface: Y-Werte
    Dim WX() As Double                                  '* Interface: X-Werte
End Structure

'###Enumerationen#############################################################
Public Enum enmChannel As Integer 'Bezeichnerstruktur zur besseren Lesbarkeit
    CH0 = 0
    CH1 = 1
    BOTH = 2
End Enum

Public Enum enmTracking As Byte 'ditto
    Phase = 0
    Frequ = 1
End Enum

Public Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'###Variablen#################################################################
Dim ProfilKurve(3) As KurvenStruktur
Dim PortName, STATUS, ADCString(3) As String
Dim ADCDec(3), MessIndex As Long
Dim ADCVolt(3), ProfilFrequenz, ProfilPhase, ProfilStart, ProfilStop, _
    ProfilStep, StartAPFrequenz, StartAPUphi, StartAPPhase, FrequenzAktuell,_
    PhaseAktuell, ProfilUGIndex, ProfilOGIndex, K, Offset, Anfangslaenge,_
    Laengenkonstante As Double
Dim Unit As String
Dim TrackingChoice As enmTracking = enmTracking.Frequ

'###Startroutinen#############################################################
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _
        System.EventArgs) Handles MyBase.Load
```

```
58      STATUS = "START"
59      FormFormat()
60      PortCheck()
61      LoadSettings()
62      Init()
63      STATUS = "RUN"
64  End Sub
65
66  Private Sub Init()
67      For Chan As Int16 = 0 To 3
68          With ProfilKurve(Chan)
69              ReDim .WX(0)
70              ReDim .WY(0)
71          End With
72      Next
73      Unit = " Hz"
74  End Sub
75
76  Private Sub FormFormat()
77
78  End Sub
79
80  Private Sub LoadSettings()
81      PortName = My.Settings.PortName
82      cboPorts.SelectedItem = PortName
83      Port1.PortName = PortName
84      Port1.Encoding = System.Text.Encoding.Default   ' in vb.net notwendig für
85          '8 Bit-Werte, falls diese Zeile fehlt, wird alles größer 127 als 63
86          'gesendet (Parity Replace)
87      Port1.Open()
88  End Sub
89
90  Private Sub PortCheck()
91      cboPorts.Items.AddRange(Port1.GetPortNames)
92  End Sub
93
94  '###Messsystemroutinen#########################################################
95  Private Sub DDS_Hz(ByVal Frequenz As Double, ByVal Channel As enmChannel)
96  'Subroutine zum Setzen der 2DDS-Frequenzen, mit Kanalauswahl und f in Hz
97      Dim XTAL As Long = 156250000 'Referenztakt der 2DDS
98      Dim WortDez As Long
99      Dim WortHex As String
100     Dim Z As Int16
101     Dim Ascii(10)
102     Dim sb As New System.Text.StringBuilder()
103     WortDez = CLng((Frequenz * (2 ^ 32)) / XTAL) 'Formel zur ↩
            Frequenzwortberechnung,
104         's. Datenblatt AD9958 Rev. A S. 18
105     WortHex = Hex(WortDez).ToString
106     WortHex = WortHex.PadLeft(8, "0") ' mit Nullen auffüllen um fehler bei ↩
            kleinen
107         'frequenzen zu entgehen
108     Port1.Write("5") '5 senden um 2DDS auszuwählen
109     If (Channel = enmChannel.CH0) Then
110         sb.Append(Convert.ToChar(5)) 'Kanalauswahl und Frequenzbit entsprechend
111             'Festlegung übertragen
112     Else
113         If (Channel = enmChannel.CH1) Then
114             sb.Append(Convert.ToChar(6))
115         Else
116             sb.Append(Convert.ToChar(7))
117         End If
```

```
118     End If
119     Port1.Write(sb.ToString)
120     Z = 4
121     'Senden Senden der Hex zeichen als Dezimalwert
122     For i = 1 To Len(WortHex) Step 2
123         Ascii(Z) = (CInt("&H" & Mid(WortHex, i, 2)))
124         Port1.Write(Chr(Ascii(Z)))
125         Z = Z - 1
126     Next
127 End Sub
128
129 Private Sub DDS_deg(ByVal Phase As Double, ByVal Channel As Integer)
130 'Subroutine zum Setzen der 2DDS-Phasenakkus, mit Kanalauswahl und phi in Grad
131     Dim XTAL As Long = 360
132     Dim WortDez As Long
133     Dim WortHex As String
134     Dim Z As Int16
135     Dim Ascii(10)
136     Dim sb As New System.Text.StringBuilder()
137     WortDez = CLng((Phase * (2 ^ 14)) / XTAL) 'Formel zur Phasenwortberechnung,
138         's. Datenblatt AD9958 Rev. A S. 18
139     WortHex = Hex(WortDez).ToString
140     WortHex = WortHex.PadLeft(8, "0") ' mit Nullen auffüllen um fehler bei
141         'kleinen frequenzen zu entgehen
142     Port1.Write("5")
143     If (Channel = enmChannel.CH0) Then 'Kanal- und Funktionsauswahl entsprechend
144         'Festlegung übertragen
145         sb.Append(Convert.ToChar(9))
146     Else
147         If (Channel = enmChannel.CH1) Then
148             sb.Append(Convert.ToChar(10))
149         Else
150             sb.Append(Convert.ToChar(11))
151         End If
152     End If
153     Port1.Write(sb.ToString)
154     Z = 4
155     'Senden Senden der Hex zeichen als Dezimalwert
156     For i = 1 To Len(WortHex) Step 2
157         Ascii(Z) = (CInt("&H" & Mid(WortHex, i, 2)))
158         Port1.Write(Chr(Ascii(Z)))
159         Z = Z - 1
160     Next
161 End Sub
162
163 Private Sub ADC974() 'Auslesen des AD974, schreibt in ADCVolt
164     Dim Chan As Int16
165     If Not Port1.IsOpen Then Port1.Open()
166     For Chan = 0 To 3
167         Port1.Write("0") : Sleep(5)
168         Port1.Write((Chan + 1).ToString("0")) : Sleep(5)
169         ADCString(Chan) = Port1.ReadExisting
170         ADCDec(Chan) = HexToDec(ADCString(Chan))
171         ADCVolt(Chan) = 5 / 65535 * ADCDec(Chan)
172     Next
173 End Sub
174
175 '###Interaktionsroutinen#################################################################
176 Private Function HexToDec(ByVal HexWert As String) As Integer
177     HexToDec = Convert.ToInt32(HexWert.TrimEnd, 16)
178     'Zeichen von Port sind mit Leerzeichen am Ende versehen
179 End Function
```

```
180
181 Private Sub cboPorts_SelectedIndexChanged(ByVal sender As System.Object, _
182         ByVal e As System.EventArgs) Handles cboPorts.SelectedIndexChanged
183     If Port1.IsOpen Then Port1.Close()
184     PortName = cboPorts.SelectedItem
185     Port1.PortName = PortName
186     My.Settings.PortName = PortName
187 End Sub
188
189 Private Sub WahlStartAP(ByVal sender As System.Object, ByVal e As _
190         System.EventArgs) Handles lbProfil.SelectedIndexChanged
191 'Auswahl des AP über Profilmesswerte in der Listbox
192     Dim Value As Integer
193     Value = lbProfil.SelectedIndex
194     If TrackingChoice = enmTracking.Frequ Then 'AP entsprechend Auswahl setzen
195         StartAPFrequenz = ProfilKurve(3).WX(Value) 'bei f-Tracking ist die
196             'Startfrequenz gleich der gewählten
197         StartAPPhase = 0 'Phase ist null
198     Else
199         StartAPFrequenz = ProfilStart 'bei Phasentracking ist AP-Freqenz gleich
200             'der Profilmessfrequenz
201         StartAPPhase = ProfilKurve(3).WX(Value) 'Anfangsphase gleich der ↩
                gewählten
202     End If
203     StartAPUphi = ProfilKurve(3).WY(Value) 'Uphi ist gleich der gewählten
204     tbFrequ.Text = StartAPFrequenz.ToString("000.00") + " Hz"
205         'optisches Feedback in den Textboxen
206     tbUphi.Text = StartAPUphi.ToString("000.000") + " V"
207     ctProfil.ChartAreas(0).CursorX.Position = ProfilKurve(3).WX(Value)
208 End Sub
209
210
211 Private Sub lbMessung_SelectedIndexChanged(ByVal sender As System.Object, _
212         ByVal e As System.EventArgs) Handles lbMessung.SelectedIndexChanged
213     ctLaenge.ChartAreas(0).CursorX.Position = lbMessung.SelectedIndex
214     ctMesskurven.ChartAreas(0).CursorX.Position = lbMessung.SelectedIndex
215 End Sub
216
217 Private Sub TrackingChange(ByVal sender As System.Object, ByVal e As _
218 System.EventArgs) Handles rbPhase.Click, rbFrequ.Click
219 'Umschalten der Trackingvarianten
220     If rbFrequ.Checked Then
221         TrackingChoice = enmTracking.Frequ
222         Label8.Visible = True
223         txtProfilStop.Visible = True
224         Label7.Text = "von"
225         Unit = " Hz"
226     ElseIf rbPhase.Checked Then
227         TrackingChoice = enmTracking.Phase
228         Label8.Visible = False
229         txtProfilStop.Visible = False
230         Label7.Text = "Frq."
231         Unit = " deg"
232     End If
233 End Sub
234
235 Private Sub UGChange(ByVal sender As System.Object, ByVal e As System.EventArgs)_
236         Handles tbarUG.Scroll
237 'Untergrenze zur Berechnung von K festlegen
238     Dim myTB As System.Windows.Forms.TrackBar
239     myTB = sender
240     ProfilUGIndex = myTB.Value
```

```
241     tbUG.Text = ProfilKurve(3).WX(ProfilUGIndex).ToString("000.00") + Unit
242     ctProfil.ChartAreas(0).CursorX.SelectionStart = ProfilKurve(3).WX(↩
            ProfilUGIndex)
243 End Sub
244
245 Private Sub OGChange(ByVal sender As System.Object, ByVal e As System.EventArgs)_
246         Handles tbarOG.Scroll
247 'Obergrenze festlegen
248     Dim myTB As System.Windows.Forms.TrackBar
249     myTB = sender
250     ProfilOGIndex = myTB.Value
251     tbOG.Text = ProfilKurve(3).WX(ProfilOGIndex).ToString("000.00") + Unit
252     ctProfil.ChartAreas(0).CursorX.SelectionEnd = ProfilKurve(3).WX(ProfilOGIndex↩
            )
253 End Sub
254
255 Private Sub btnCalcK_Click(ByVal sender As System.Object, ByVal e As _
256         System.EventArgs) Handles btnCalcK.Click
257 'Anstieg K berechnen
258     Dim Grenze, i As Integer
259     Grenze = ProfilOGIndex - ProfilUGIndex
260     Dim Data(1, Grenze) As Double 'zweidimensionales Array mit (x,y)-Wertepaaren
261     Dim xmittel As Double = 0, ymittel As Double = 0 'Mittelwerte
262     Dim Sum1 As Double = 0, Sum2 As Double = 0 'Summen
263
264     'lineare Regression um Gerade zu ermitteln
265     For i = ProfilUGIndex To ProfilOGIndex
266         'Auslesen der (x,y)-Werte in eigene Datestruktur, Summenbildung zu ↩
                Mittelwertberechnung
267         Data(0, i - ProfilUGIndex) = ProfilKurve(3).WX(i) / 1000000 '↩
                Skalierungsfaktor, da bei f-tracking der Anstieg sehr gering in V/Hz
268         Data(1, i - ProfilUGIndex) = ProfilKurve(3).WY(i)
269         xmittel += Data(0, i - ProfilUGIndex)
270         ymittel += Data(1, i - ProfilUGIndex)
271     Next
272     'Mittelwertberechnung
273     xmittel /= (Grenze + 1)
274     ymittel /= (Grenze + 1)
275     'Summen bilden
276     For i = 0 To Grenze
277         Sum1 += (Data(0, i) - xmittel) * (Data(1, i) - ymittel)
278         Sum2 += (Data(0, i) - xmittel) * (Data(0, i) - ymittel)
279     Next
280     'K berechnen
281     K = Sum1 / Sum2
282     'Offset berechnen, eigtl nicht benötigt, höchstens zur grafischen Darstellung
283     Offset = ymittel - K * xmittel
284
285     If TrackingChoice = enmTracking.Phase Then
286         K /= 1000000 'wenn Phasentracking Skalierung zurückrechnen
287         tbK.Text = K.ToString("0.0000") + " deg/V"
288     Else
289         tbK.Text = K.ToString("0.0000") + " MHz/V"
290     End If
291 End Sub
292
293 Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal _
294         e As System.EventArgs) Handles ComboBox1.SelectedIndexChanged
295     '???
296 End Sub
297
298 Private Sub btnFile_Click(ByVal sender As System.Object, ByVal e As _
```

```
299         System.EventArgs) Handles btnFile.Click
300 'Messwerte in txt-File ablegen
301     Dim sbPath As New System.Text.StringBuilder
302     Dim datDatum As Date
303     Dim D As DirectoryInfo
304     datDatum = Now
305     sbPath.Length = 0
306     sbPath.Append(Path.GetDirectoryName(Application.ExecutablePath))
307     sbPath.Append("\Messung\ ")
308     If TrackingChoice = enmTracking.Frequ Then
309         sbPath.Append("Frequency\ ")
310     Else
311         sbPath.Append("Phase\ ")
312     End If
313     D = New DirectoryInfo(sbPath.ToString)
314     If Not D.Exists Then
315         D.Create()
316     End If
317     sbPath.Append(datDatum.ToString("yyyy.MM.dd"))
318     sbPath.Append(" - ")
319     sbPath.Append(datDatum.ToString("HHmmss"))
320     sbPath.Append(".txt")
321     Dim fs As StreamWriter = New StreamWriter(sbPath.ToString, FileMode.CreateNew↩
            )
322     fs.WriteLine("fAP: " + StartAPFrequenz.ToString("0.000") + " Hz; phiAP: " + ↩
            StartAPPhase.ToString("0.000") + _
323                  " deg; Uphi: " + StartAPUphi.ToString("0.00000") + " V; l: " + ↩
                       Anfangslaenge.ToString("0.00"))
324     For i As Integer = 0 To lbMessung.Items.Count - 1
325         fs.WriteLine(lbMessung.Items.Item(i).ToString)
326     Next
327     fs.Close()
328 End Sub
329
330 Private Sub btnSaveProfile_Click(ByVal sender As System.Object, ByVal e As _
331         System.EventArgs) Handles btnSaveProfile.Click
332     Dim sbPath, sbLine As New System.Text.StringBuilder
333     Dim datDatum As Date
334     Dim D As DirectoryInfo
335     datDatum = Now
336     sbPath.Length = 0
337     sbPath.Append(Path.GetDirectoryName(Application.ExecutablePath))
338     sbPath.Append("\Pofile\ ")
339     If TrackingChoice = enmTracking.Frequ Then
340         sbPath.Append("Frequency\ ")
341     Else
342         sbPath.Append("Phase\ ")
343     End If
344     D = New DirectoryInfo(sbPath.ToString)
345     If Not D.Exists Then
346         D.Create()
347     End If
348     sbPath.Append(datDatum.ToString("yyyy.MM.dd"))
349     sbPath.Append(" - ")
350     sbPath.Append(datDatum.ToString("HHmmss"))
351     sbPath.Append(".txt")
352     Dim fs As StreamWriter = New StreamWriter(sbPath.ToString, FileMode.CreateNew↩
            )
353     fs.WriteLine("K: " + K.ToString("0.000000") + " OG: " + _
354                  ProfilOGIndex.ToString() + " UG: " + ProfilUGIndex.ToString())
355     For i As Integer = 0 To lbProfil.Items.Count - 1
356         fs.WriteLine(lbProfil.Items.Item(i).ToString)
```

```
357     Next
358     fs.Close()
359 End Sub
360
361 '###Messroutinen#########################################################################
362 Private Sub btnStart_Click(ByVal sender As System.Object, ByVal e As _
363         System.EventArgs) Handles btnStart.Click
364 'Start der Längenüberwachung
365     Dim temp As Double = 0
366     lbProfil.Enabled = False 'Menüelemente deaktivieren
367     rbFrequ.Enabled = False
368     rbPhase.Enabled = False
369     lbMessung.Items.Clear()
370     DDS_Hz(StartAPFrequenz, enmChannel.BOTH) 'Anfangs-/Arbeitstpunktfrequenz ↩
            setzen
371     DDS_deg(StartAPPhase, enmChannel.CH0) 'Anfangs-/Arbeitspunktphase setzen
372     PhaseAktuell = StartAPPhase
373     FrequenzAktuell = StartAPFrequenz
374     For Chan As Int16 = 0 To 3
375         ctMesskurven.Series(Chan).Points.Clear()
376     Next
377     MessIndex = 0
378     ctLaenge.Series(0).Points.Clear()
379     ctLaenge.Series(1).Points.Clear()
380     Anfangslaenge = CType(tbLaenge.Text, Double) 'Anfangslänge einlesen
381     If TrackingChoice = enmTracking.Frequ Then 'Längenkonstante berechnen
382         Laengenkonstante = Anfangslaenge * FrequenzAktuell
383     Else
384         Laengenkonstante = Anfangslaenge / PhaseAktuell
385     End If
386     timer_adc.Start() ' Timer ADC auslesen starten
387 End Sub
388
389 Private Sub timer_adc_Tick(ByVal sender As System.Object, ByVal e As _
390         System.EventArgs) Handles timer_adc.Tick
391 'Messwerterfassung
392     Dim SB As New System.Text.StringBuilder
393     Dim tmp As Double
394     SB.Length = 0
395
396     tmp = 0
397     For i = 1 To 10
398         ADC974() '*Werte vom ADC einlesen
399         tmp += ADCVolt(3)
400     Next
401     tmp /= 10
402     ADCVolt(3) = tmp
403
404     'Regelung
405     If TrackingChoice = enmTracking.Frequ Then
406         FrequenzAktuell = FrequenzAktuell + ((StartAPUphi - ADCVolt(3)) / _
407             (K / 1000000)) 'neue Frequenz errechnen aus Uphi und Regelkonstante
408         DDS_Hz(FrequenzAktuell, enmChannel.BOTH) 'Frequenz setzen
409         Anfangslaenge = Laengenkonstante / FrequenzAktuell
410         'neue Länge aus aktueller Frequenz und Längenkonstante ermitteln
411         'Anfangslaenge = (49965410 * (1 / StartAPFrequenz + 1 / FrequenzAktuell))
412     Else
413         PhaseAktuell = PhaseAktuell + ((StartAPUphi - ADCVolt(3)) / (K))
414         'neue Phase aus gemessenem Uphi und Regelkonstante errechnen
415         If PhaseAktuell < 0 Then
416             Do Until (PhaseAktuell >= 0)
417                 PhaseAktuell += 360
```

```
418             Loop
419         ElseIf PhaseAktuell > 360 Then
420             Do Until (PhaseAktuell <= 360)
421                 PhaseAktuell += 360
422             Loop
423         End If
424         DDS_deg(PhaseAktuell, enmChannel.CH0) 'Phase setzen
425         Anfangslaenge = Laengenkonstante * PhaseAktuell 'neue Länge ermitteln
426     End If
427
428     'Messwerte ablegen
429     SB.Append(MessIndex.ToString("00000") & " ")
430     ctLaenge.Series(0).Points.AddXY(MessIndex, Anfangslaenge)
431     If TrackingChoice = enmTracking.Frequ Then
432         ctLaenge.Series(1).Points.AddXY(MessIndex, FrequenzAktuell)
433     Else
434         ctLaenge.Series(1).Points.AddXY(MessIndex, PhaseAktuell)
435     End If
436     SB.Append(Anfangslaenge.ToString("000.000" & " "))
437     SB.Append(FrequenzAktuell.ToString("00000000.0" & " Hz "))
438     SB.Append(PhaseAktuell.ToString("000.000" & " deg "))
439     For Chan As Int16 = 0 To 3
440         ctMesskurven.Series(Chan).Points.AddXY(MessIndex, ADCVolt(Chan))
441         SB.Append(ADCVolt(Chan).ToString("0.00000") & " V ")
442     Next
443     lbMessung.Items.Add(SB.ToString)
444     MessIndex += 1
445 End Sub
446
447 Private Sub btnStop_Click(ByVal sender As System.Object, ByVal e As _
448         System.EventArgs) Handles btnStop.Click
449 'Timer anhalten, Menüelemente reaktivieren
450     timer_adc.Stop()
451     lbProfil.Enabled = True
452     rbFrequ.Enabled = True
453     rbPhase.Enabled = True
454 End Sub
455
456 Private Sub btnEnd_Click(ByVal sender As System.Object, ByVal e As _
457         System.EventArgs) Handles btnEnd.Click
458 'Programm beenden
459     Me.Close()
460 End Sub
461
462 Private Sub btnStartProfil_Click(ByVal sender As System.Object, ByVal e As _
463         System.EventArgs) Handles btnStartProfil.Click
464 'Profilmessung starten
465     '*txtProfil.Text = ""
466     lbProfil.Items.Clear()
467     MessIndex = 0
468     tbarOG.Value = 0 'Auswahlregler für Ober-/Untergrenze auf null setzen um
469     'Out-of-Range-Fehler beim Picturebox zeichnen zu vermeiden
470     tbarUG.Value = 0
471
472     For Chan As Int16 = 0 To 3
473         ReDim ProfilKurve(Chan).WY(0)
474         ReDim ProfilKurve(Chan).WX(0)
475         ctProfil.Series(Chan).Points.Clear()
476     Next
477     Try
478         ProfilStart = CType(txtProfilStart.Text, Double)
479         'ist bei Frequenztracking Startfrequenz, bei Phasentracking AP-Frequenz
```

```
480         ProfilStop = CType(txtProfilStop.Text, Double)
481         'f-Tracking: Endfrequenz; phi-Tracking: ungenutzt
482         ProfilStep = CType(txtProfilStep.Text, Double)
483         'Schrittweite in entweder Hz oder Grad
484         ProfilFrequenz = ProfilStart
485         ProfilPhase = 0
486         DDS_deg(0, enmChannel.BOTH)
487         DDS_deg(180, enmChannel.CH1)
488         tmrProfil.Start() 'Timer starten
489     Catch ex As Exception
490         MsgBox("Eingabefehler")
491     End Try
492 End Sub
493
494 Private Sub tmrProfil_Tick(ByVal sender As System.Object, ByVal e As _
495         System.EventArgs) Handles tmrProfil.Tick
496 'Profilmesswerte aufnehmen
497     Dim SB As New System.Text.StringBuilder
498     SB.Length = 0
499     DDS_Hz(ProfilFrequenz, enmChannel.BOTH) 'aktuelle Werte setzen
500     DDS_deg(ProfilPhase, enmChannel.CH0)
501     ADC974() 'AD974 auslesen
502     SB.Append(ProfilFrequenz.ToString("00000000.0") & " Hz " & _
503               ProfilPhase.ToString("000.000") & " deg ")
504
505     '* Erfassen der Messwerte
506     For Chan As Int16 = 0 To 3
507         ReDim Preserve ProfilKurve(Chan).WY(MessIndex)
508         ReDim Preserve ProfilKurve(Chan).WX(MessIndex)
509         ProfilKurve(Chan).WY(MessIndex) = ADCVolt(Chan)
510         If TrackingChoice = enmTracking.Frequ Then
511             ProfilKurve(Chan).WX(MessIndex) = ProfilFrequenz
512             ctProfil.Series(Chan).Points.AddXY(ProfilFrequenz, ADCVolt(Chan))
513         Else
514             ProfilKurve(Chan).WX(MessIndex) = ProfilPhase
515             ctProfil.Series(Chan).Points.AddXY(ProfilPhase, ADCVolt(Chan))
516         End If
517         SB.Append(ADCVolt(Chan).ToString("0.00000") & " V  ")
518     Next
519     lbProfil.Items.Add(SB.ToString) 'Messwerte in Listbox eintragen
520     tbarOG.Maximum = MessIndex 'Auswahlregler für Ober-/Untergrenze erweitern
521     tbarUG.Maximum = MessIndex
522     If TrackingChoice = enmTracking.Frequ Then
523     'je nach Trackingvariante Frequenz oder Phase um Stepweite erhöhen
524         ProfilFrequenz += ProfilStep
525         If ProfilFrequenz > ProfilStop + 1 Then
526         'beim erreichen der Obergrenze Messungsende
527             tmrProfil.Stop()
528         Else
529             MessIndex += 1
530         End If
531     Else
532         ProfilPhase += ProfilStep
533         If ProfilPhase > 360 Then
534             tmrProfil.Stop()
535         Else
536             MessIndex += 1
537         End If
538     End If
539 End Sub
540 End Class
```

Literaturverzeichnis

[1] *FAQ AD9958*. Website. Online verfügbar unter `http://www.analog.com/en/rfif-components/direct-digital-synthesis-dds/ad9958/products/faqs/resources.html?display=popup`; aufgerufen am 23. November 2010.

[2] *Wikipedia Artikel Cauer-Filter*. Website. Online verfügbar unter `http://de.wikipedia.org/wiki/Elliptische_Filter`; aufgerufen am 20. Januar 2011.

[3] Analog Devices Inc.: *A Technical Tutorial on Digital Signal Synthesis*, 1999. Online verfügbar unter `http://www.analog.com/static/imported-files/tutorials/450968421DDS_Tutorial_rev12-2-99.pdf`; aufgerufen am 3. Dezember 2010.

[4] Analog Devices Inc.: *Datenblatt AD8302 Rev. A*, 2002. Online verfügbar unter `http://www.analog.com/static/imported-files/data_sheets/AD8302.pdf`; aufgerufen am 21. November 2010.

[5] Analog Devices Inc.: *Datenblatt AD9958 Rev. A*, 2008. Online verfügbar unter `http://www.analog.com/static/imported-files/data_sheets/AD9958.pdf`; aufgerufen am 10. Dezember 2010.

[6] Atmel Corporation: *Datenblatt ATmega128 Rev. U*, 2009. Online verfügbar unter `http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf`; aufgerufen am 10. Dezember 2010.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Leipzig, 18. Februar 2011 _____